LA-13925

Approved for public release;
distribution is unlimited.

EOLUS Project and Monte Carlo Team Software Quality Assurance Plan

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

Edited by Vin LoPresti, Group IM-1

An Affirmative Action/Equal Opportunity Employer

LA-13925

Issued: April 2002

EOLUS Project and Monte Carlo Team Software Quality Assurance Plan

Gregg Giesler

Los Alamos NM 87545

TABLE OF CONTENTS

LIST OF FIGURES

EOLUS PROJECT AND MONTE CARLO TEAM

SOFTWARE QUALITY ASSURANCE PLAN

by

Gregg Giesler

ABSTRACT (U)

MCNP™ is a computer code that is used to describe the transport of neutrons, photons, and electrons through matter. This Software Quality Assurance Plan is a revision of the previously published plan and describes the process to be used by the Eolus Project and Monte Carlo team to improve software quality while maintaining and enhancing this and other codes. The software development process and its implementation in the Razor™ tool are described in detail. Included in this document is the Software Verification and Validation Plan.

™ MCNP is a trademark of the Regents of the University of California, Los Alamos National Laboratory.
™ Razor is a trademark of Tower Concepts, now a part of Visible Systems Corporation.

I. PURPOSE

The X-5 Monte Carlo team is responsible for developing and maintaining computer codes that simulate particle and radiation transport using the Monte Carlo method. Two of its principal codes are MCNP (Monte Carlo N-Particle), a radiation transport code based on codes developed and maintained at the Los Alamos National Laboratory (LANL) for more than fifty years.[1,2] and LAHET,[3] a high-energy particle transport code based on HETC[4] from Oak Ridge National Laboratory. Both are mature products that are continually being upgraded with a major version released for international distribution every few years; and MCNP has approximately quarterly intermediate releases of newly integrated features for internal use and testing. Both are supported on a variety of computer platforms, primarily in the UNIX and Windows operating environments. The Eolus Project of the Advanced Simulation and Computing (ASCI) Program is the current program providing most of the financial support for this team.

In 1996, a Software Quality Assurance (SQA) Plan[5] was written for MCNP because one of its authors thought it was important to document the process used to maintain MCNP. The plan described how new features and bug fixes were developed and reviewed to provide code for implementation. It also described how intermediate and major releases were created, reviewed, and released. This document replaces that report.

In 1999, the Monte Carlo team decided to implement a software-based, auditable version of this process and apply it to all of its code products: MCNP, LAHET, and others. There were several reasons for this decision. The foremost is that the ASCI Program required improvements in the way its projects developed software. Using a software-based tool ensures that a consistent development process is followed. This places more responsibility for the quality of the code on the individual developer. By following a defined process with intermediate reviews, the developer will have to more clearly define what is to be done and how it is done *before* the coding is completed. As a result, the integration of the new code into the base code should be much easier, and fewer failures in system testing should result. Also, this should enhance software development skills of members of the team and improved performance by the team. Most important, this should result in improved software products.

Also, a software-based tool makes it much easier to audit the software development process. As part of formalizing the process, a tool can track who moved an issue to the next

step and when the step was started. It also provides the ability to keep a log of what was done during each step of a process. Additional capabilities of configuration control and release management that can also be linked to issues in the development process will make the process more consistent and auditable. Using a software-based tool removes the need to handle and track mountains of paper that would be needed to document such a development process manually.

The purpose of this SQA Plan is to describe the process the Monte Carlo team uses to maintain and improve the quality of the codes developed and maintained by its members. These currently include MCNP Version 4C3 and LAHET Version 3.16. MCNP is used to describe the transport of neutrons, photons, electrons, and other particles through matter and is used for many purposes including radiation shielding, criticality safety, oil well logging, and medical imaging and treatment calculations, to mention a few. LAHET is used in the accelerator, medical, and space sciences communities for facility, detector, and shielding design. MCNP has an international user community of over 3,000 users at more than 260 institutions, while LAHET has an international user community of about 200.

The scope of this SQA plan is the ongoing maintenance phase of a typical software life cycle of these codes. This phase includes change proposal, requirements and design, implementation, test, installation and checkout, release, and maintenance for new features, enhancements, and bug fixes to the codes. This plan is to replace the previous SQA plan and is being written to reflect the use of a software-based tool to control and monitor this software development process. The process itself and the implementation in Razor, a tool that combines issue tracking, version control, and release management, are described in detail in Reference 6 that is extracted into Appendices B through F. Additional documents will be produced providing additional details for some of the process steps. This plan does not cover the quality assurance of the data libraries distributed and used with these codes.

This SQA plan is written primarily for the Monte Carlo team members who maintain and improve these codes. However, the audience of this SQA plan is not only the members of the Monte Carlo team, but also its international user community and the managers of the ASCI Program of which the Eolus Project is currently a part. This document is an indication of how the codes are maintained and improved and will be used as a reference in audits by organizations external to this team.

This plan has seven appendices. Their contents are as follows.

- Appendix A—definitions and abbreviations
- Appendix B—a narrative description of the Monte Carlo software development process
- Appendix C—a description of the implementation of this process in Razor
- Appendix D—detailed description of each step in each subprocess
- Appendix E—the activities assigned to each role mentioned in Appendix D
- Appendix F—a description of the work products generated from the subprocesses detailed in Appendix D
- Appendix G—Software Verification and Validation Plan

This plan is compliant with IEEE Std 730-1998[7] and IEEE Std 730.1-1995.[8] Also, ISO 9001:2000[9] and ISO 9000-3:1997[10] were used in preparation of this plan.

II. MANAGEMENT

A. Organization

Software quality assurance for these codes is the responsibility of the Monte Carlo team itself. This team, which is lead by a Team Leader (a nonmanagement position at LANL), is at this writing, part of the Diagnostics Applications Group (X-5) of the Applied Physics (X) Division of LANL.

The team is also part of and is currently primarily supported by the ASCI Program. The ASCI Program is a program of the National Nuclear Security Administration (NNSA) of the U.S. Department of Energy (DOE) that includes the three DOE nuclear weapons laboratories. Within each laboratory, its projects include part or all of some line organizations. As a result, the Monte Carlo team leader as the Eolus Project leader also reports to the LANL office for the ASCI Program. Although neither the higher-level LANL organizations nor the ASCI Program have a documented SQA plan in place, these plans are being developed. DOE has recently issued a Software Quality Assurance notice,[11] ASCI has drafted a document on Software Quality Engineering,[12] and a working group in the LANL ASCI Program has prepared a recommendation for a SQA plan[13] for the LANL organizations.

A Board of Directors, composed of team members, other group members, and members of other teams, as appropriate, reviews all proposed new features and their implementation,

especially changes to the user interface. The Board of Directors meetings are held on an as needed basis.

B. Tasks

A summary of the tasks to be performed as part of the Monte Carlo team software development includes change proposal, requirements and design, implementation, test, installation and checkout, release, and maintenance for new features, enhancements, and bug fixes to the codes. Releases can be intermediate releases of new features and enhancements for testing and use by the local user community, and major releases to the international user community. Several reviews occur during the execution of these tasks. A narrative description of this software development process is found in Appendix B, a description of the implementation in Razor is found in Appendix C, and a detailed description of each step within the process is found in Appendix D.

This plan shall be updated as the software development process evolves. After review and approval by the team and the Board of Directors, the updated plan shall be distributed and archived in the configuration management database.

C. Responsibilities

Software quality assurance activities are performed primarily by members of the Monte Carlo team. In addition to developing code, team members perform the reviews, and designated team members perform the testing and integration of new code into the baseline. Designated members of the team will oversee the performance of the SQA activities. Detailed responsibilities for each of the roles in this development process are found in Appendix E.

The maintenance of this SQA plan shall be done by members of the Monte Carlo team. Suggestions for revisions shall be submitted as issues and worked like all other issues. The full team shall review and approve the plan with final authority resting with the team leader. The plan shall be distributed to members of the team and as determined by the team leader.

III. DOCUMENTATION

The work products required by this software development process are described in Appendix F. These items are some of the outputs of the individual steps of the subprocesses detailed in Appendix D. The work products produced by these subprocesses include Software Requirements Specifications, Software Design Descriptions, Code, Software Documentation,

Software Test Plans and Software Test Results, and Review Results. In Appendix F, each of these products is described, where it is created, used, and reviewed is detailed, and references to relevant standards are included.

IV. STANDARDS, PRACTICES, CONVENTIONS, AND METRICS

The ANSI C[14] and Fortran 90/95[15] standards are to be used, and the proposed modifications to those standards shall be considered for all new code developments and modifications. All code is compiled with ANSI C and Fortran standard compilers, thus assuring language standard compatibility. Compliance with Fortran 77[16] will no longer be required, especially for features listed as obsolete or deleted in the Fortran 90/95 standard and to be deleted in the proposed Fortran standard under development. Also, Fortran 77 standard compilers are rapidly losing support.

Proposed coding and file documentation and commentary standards are found in Ref. 17. Use of these standards is encouraged and will be required as the team adopts these standards.

Metrics that will be collected will be determined during the implementation and use of Razor and by ASCI Program management. The metrics collected will be used as a measure of the software quality of the products produced, as a means to determine process improvements, or as reportable items to management. Details as to which metrics will be collected are awaiting the completion and publication of the LANL ASCI Software Quality Engineering document.

V. REVIEWS AND AUDITS

The reviews and audits conducted by the Monte Carlo team are described in Appendices B and D. Additional reviews and audits may be conducted by higher-level organizations as they are determined to be necessary. Included in this is an annual review by ASCI of all of its projects.

Internal reviews of compliance of the team with the process will also be conducted. These reviews will be used to evaluate the process in order to improve the process and reduce defects in the products.

VI. TEST

Over the years, a number of test suites have been developed for these codes. A regression test set containing more than thirty problems has been developed as one form of verification for MCNP. This test set is used to verify that any changes to the code have not adversely affected the code and is continually upgraded as changes are made to the code. It is also being used as an installation test set. This test set is described in Ref. 18 that was updated in Ref. 19. Additional tests have been developed and are run to better characterize specific changes to or features of the code. Similar test sets have been developed for LAHET and other Monte Carlo team codes.

In addition, a number of benchmark calculations have been performed with MCNP and its LANL databases as validation to ensure agreement with physical measurements. The benchmarking program is an ongoing and collaborative effort, and a number of benchmarking documents are available from LANL and outside organizations that have done MCNP benchmarking. A list of LANL benchmark publications is available from the team. Additionally, many other laboratories throughout the world benchmark MCNP for their own purposes, and the scientific literature includes approximately 10–30 MCNP benchmarks per year by other organizations. These are not part of this SQA Plan because they are primarily conducted by the other organizations for their own purposes, but they do add additional confidence in the code and the associated LANL databases.

A description of the test sets used by this team is found in the Software Test Plan currently under development.

VII. PROBLEM REPORTING AND CORRECTIVE ACTION

All suspected problems with these Monte Carlo codes should be reported to the Monte Carlo team. The e-mail address for MCNP problems is mcnp@lanl.gov. Problems with LAHET can be reported to lcs-forum@lanl.gov, and problems with other codes can be reported to the developer or the team leader. The goal is to enter all reported problems into the issue-tracking database, and to have the team leader examine each entry. Those that are not user error will be assigned to a developer for investigation and correction. Trivial bugs (incorrect spellings in the code, etc.) can be quickly and easily processed and documented. The defect reports will be made available to users on the Eolus website.

Corrective action will be entered into the issue-tracking database as part of the issue. The corrected code will be included in the next release of the code. A separate memo may also be prepared describing the problem fix, consequences, and any possible work-around. This memo will be referenced in the issue tracking, file tracking, and release tracking. Some of these corrections may be posted to the website prior to the next release of the code.

Problems identified in the software development and maintenance process are the responsibility of the Monte Carlo team leader who is ultimately responsible for software problem reporting and corrective action. As stated above, they should be reported as issues in the issue-tracking tool.

VIII. TOOLS, TECHNIQUES, AND METHODOLOGIES

Razor,[20] a product of Visible Systems Corporation, is the tool currently being used by the Monte Carlo team for problem tracking, version control, and release management. All proposed new features and enhancements and reported defects in the released code are entered into its problem-tracking tool. The development process for each of these items, including the reviews, is then tracked through the tool through release and closeout. The version control tool contains not only the source code, but also all documentation and other items related to the codes and to the process itself. The release management tool is used to track intermediate and major releases.

Another tool being used is GNU-Make, an open source software tool. It is being used as the basis for the build system being for the Fortran 90/95 versions of the codes.

IX. CODE CONTROL

Code control is maintained by access controls in the version-control tool. Team members are required to branch any locked files that they may want to modify. Only the Integrator role is allowed to merge changes into locked (trunk) files.

Intermediate releases are made available to local users through the Los Alamos file system. Major releases of MCNP and LAHET are released through the Radiation Safety Information Computational Center (RSICC) in Oak Ridge, TN.

X. MEDIA CONTROL

All development and archive files will be maintained in the development database, which is backed up into LANL computer archives. Currently, no media are being released by the Monte Carlo team, so none have to be controlled.

Major releases of the software are made available to users through the RSICC, on RSICC media, according to RSICC procedures.

XI. SUPPLIER CONTROL

The only specific product currently used by this project is the Razor software from Visible Systems Corporation. It is installed and used on many systems within LANL. This software is tested, especially for security problems, before new versions are installed on the production systems used by this project.

MCNP, LAHET, and the other codes have the usual reliance on the computer environment in which they are used. This includes the use of standard ANSI Fortran and C compilers and math libraries and graphics libraries (X Windows,[21] Compaq Visual Fortran QuickWin,[22] Lahey/Fujitsu Fortran Winteracter[23]). The multiprocessing capabilities also depend on distributed processor multiprocessing software and libraries such as OpenMP[24] for shared memory multiprocessing (SMMP) and PVM,[25] UPS,[26] and MPI[27] for distributed memory multiprocessing (DMMP). Other software may also be needed to support these capabilities. Being part of the LANL computing environment, the Monte Carlo team has no control over these, other than to report problems discovered to the appropriate support organization.

The Monte Carlo team may contract with other individuals and organizations within or outside of LANL for development of specified features and enhancements. Although the LANL business operations would handle the mechanics of the contracting, the team, and especially the team leader, will be responsible for defining the scope of the work and overseeing its execution and completion. It is desired that such work would be performed in accordance with this SQA Plan, and the team would be responsible for controlling the integration of any contracted code into the baseline.

XII. RECORDS COLLECTION, MAINTENANCE, AND RETENTION

All records developed under this plan shall be available to all members of the team through the configuration management process stated in the appendices. They will be retained there at least for the life of the project and will be transferred to other tools as they are adopted. Access to documents resulting from the working of the issues shall be made to others, as appropriate, upon request. As stated in the detailed process descriptions, approved changes to the codes shall be released to the using public.

XIII. TRAINING

The Monte Carlo team presents several MCNP training courses each year at various sites throughout the country. Both the Basic User Course and the Advanced User Course are taught, but at different times and places. These courses also provide ample opportunity for new team members to improve their skills in using MCNP.

In May 2000, all team members were trained on using Razor and the team's software development process as implemented in Razor. The slides from this course have been edited and are available in the database as is the document describing the process and implementation. These may be extracted at any time by any member of the team for training or review. These will also be used as part of the training that new members receive when they join the team.

Training opportunities in other areas of software development and software engineering can be made available to the Monte Carlo team, higher management levels, and other organizations within LANL. Training opportunities external to LANL can also be utilized.

XIV. RISK MANAGEMENT

The codes developed and maintained by the Monte Carlo team are not commercial codes and are not as constrained to specific cost and schedule restraints as commercial codes are. However, there are risks associated with the project software development activities. These risks can be divided into two types: external and internal.

External risks can arise from any management level not directly involved with project management details. Although none of the codes have specific cost or schedule constraints that commercial development activities have for the addition of new features and enhancements and the elimination of defects, funding and timing constraints are present. This project does not

have an unlimited budget. Therefore, the level of effort for development activities is limited. Also, higher management levels may impose milestones for the implementation of specific new features. Meeting one of these milestones may result in tradeoffs being made in the development of other new features and enhancements and defect removal.

Another large risk is that other commitments of LANL, such as support for other codes, may become a significant risk to completion of activities related to the team codes by the assignment of one or more team members to activities related to those other commitments.

Internal risks arise from the direct staffing of this project. The technical quality of the codes is dependent on the expertise of the team members. Acquiring and retaining qualified team members is always a problem. This can be affected by the restrictions from X Division and LANL as well as by the environment external to X Division and LANL. The maintenance and advancement of the expertise of team members can be performed by use of the opportunities described in the training section above.

These risks can lead to numerous types of deficiencies. Code changes may not be implemented in a timely manner. External and internal documentation may be inadequate or missing entirely, and may not be of adequate quality. The test set may not adequately test all parts of the code. The results of these deficiencies can not only lead to incorrect results for uses within the capabilities of the codes, but also to use of the codes for applications for which they are not suitable. A formal risk assessment and management document addressing the risks and mitigation activities for them has not been written for this project.

REFERENCES

1. Judith F. Briesmeister, ed., "MCNP—A General Monte Carlo N-Particle Transport Code, Version 4C," Los Alamos National Laboratory Report LA–13709-M (March 2000).
2. Gregg C. Giesler, "MCNP Software Quality: Then and Now," Proceedings of 10th International Conference on Software Quality, American Society for Quality, 611 East Wisconsin Avenue, Milwaukee, WI 53202 (October 2000).
3. Richard E. Prael and Henry Lichtenstein, "User Guide to LCS: The LAHET Code System," Los Alamos National Laboratory Report LA-UR-89-3014 (September 1989).
4. Radiation Shielding Information Center, "HETC Monte Carlo High-Energy Nucleon-Meson Transport Code," Oak Ridge National Laboratory Report CCC-178 (August 1977).
5. Hilary M. Abhold and John S. Hendricks, "MCNP™ Software Quality Assurance Plan," Los Alamos National Laboratory Report LA–13138 (April 1996).
6. Gregg C. Giesler, "Eolus Software Development Process and Its Implementation in Razor™," Los Alamos National Laboratory document LA–UR–00–1437 (April 2000).
7. IEEE Std 730-1998, "IEEE Standard for Quality Assurance Plans," Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017-2394 (June 25, 1998).
8. IEEE Std 730.1-1995, "IEEE Guide for Quality Assurance Planning," Institute of Electrical and Electronic Engineers, Inc., 345 East 47th Street, New York, NY 10017-2394 (December 12, 1995).
9. ISO 9001:2000, "Quality Management Systems—Requirements," ISO (International Organization for Standards) Central Secretariat, Case postale 56, CH-1211 Geneve 20, Switzerland (2000).
10. 10.ISO 9001:2000, "Quality Management and Quality Assurance Standards—Part3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software," ISO (International Organization for Standards) Central Secretariat, Case postale 56, CH-1211 Geneve 20, Switzerland (1997).
11. US DOE Notice 203.1 "Software Quality Assurance," (October 20, 2000).

12. LANL ASCI SQE Working Group, "ASCI-SQE Working Group Recommendations for LANL ASCI Software Engineering Requirements," Los Alamos National Laboratory Report LA_UR-01-6793 (December 2001).
13. Ann Hodges, et. al., "ASCI Software Quality Engineering: Goals, Principles, and Guidelines," DOE Document DOE/DP/ASC-SQE-2000-FDRFT-VERS2 (February 2001).
14. ANSI/ISO/IEC 9899-1999 "Programming Languages - C," American National Standards Institute, 1819 L Street NW, Washington, D.C. 20036 (1999).
15. ANSI X3.198-1992 (R1997) "Programming Language - Fortran-Extended," American National Standards Institute, 1819 L Street NW, Washington, D.C. 20036 (1997).
16. ANSI X3.9-1978 "Programming Language—Fortran," American National Standards Institute, 1819 L Street NW, Washington, D.C. 20036 (1978).
17. Lawrence J. Cox, "Standards For Writing and Documenting Fortran 90 Code," Los Alamos National Laboratory document X–5–RN(U)00-28 (unpublished).
18. Ronald C. Brockhoff and John S. Hendricks, "A New MCNP™ Test Suite," Los Alamos National Laboratory Report LA–12839 (September 1994).
19. John S. Hendricks, and John D. Court, "MCNP4B™ Verification and Validation," LosAlamos National Laboratory Report LA–13181 (August 1996).
20. "Razor™ Release Management, File Version Control, Problem Tracking," Tower Concepts, 248 Main Street, Oneida , NY 13421 (April 1999).
21. http://www.x.org
22. http://www.compaq.com/fortran
23. http://www.lahey.com
24. http://www.openmp.org
25. G. A. Geist, et. al., "PVM 3.0 User's Guide and Reference Manual," Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, TN (1993).
26. Richard Barrett and Mike McKay Jr., "Unified Parallel Software User's Guide and Reference Manual," http://laurel.lanl.gov:80/XCI/PROJECTS/UPS/doc/UserGuideHTML/ (2001).
27. Message Passing Interface Forum, "MPI: A message-passing interface standard," International Journal of Supercomputing Applications **8** (1994).

APPENDIX A.

DEFINITIONS AND ACRONYMS

ASCI	Advanced Strategic Computing Initiative. A program sponsored by the National Nuclear Security Administration of the U.S. Department of Energy.
Base code or baseline	The latest completed integration version of a code. It may include changes after the latest Intermediate or Major Release, which in turn, are identified by threads.
Benchmark	(1) A standard against which measurements or comparisons can be made. (2) A procedure, problem, or test that can be used to compare systems or components to each other or to a standard as in (1) (IEEE Std. 610.12-1990).
Board of Directors (BoD)	A representative advisement group from the Eolus user community including at least one nonteam member interested in the items being reviewed. The board provides advice on the development and completion of new features and the major release of code versions. The team leader or ASCI Project Office may override the recommendations of the board.
Contributor	A person not on the Monte Carlo team that provides code to be included in a team product.
Developer	A person with access to the version control tool who produces a change for a code baseline. It is also a role performing that function in the development tools.
DOE	Department of Energy
Eolus Project	A LANL code project within ASCI
IEEE	Institute of Electrical and Electronics Engineers
Integrator	A Monte Carlo team member who integrates a change into the code baseline. It is also a role performing that function in the development tools.
Intermediate Release	A revision of a major version that includes new features, but has not been released for international distribution.

ISO	Organisation Internationale de Normalisation (International Organization for Standardization).
LAHET	Los Alamos High Energy Transport, a particle transport code based on HETC from Oak Ridge National Laboratory.
LANL	Los Alamos National Laboratory
Major Release	A version released to RSICC for international distribution.
MCNP	Monte Carlo N-Particle; a radiation transport code which uses the Monte Carlo method and is designed to transport neutrons, electrons, and gamma rays in the energy range up to at least 20 MeV and to as high as 1 GeV, depending on the data in the cross-section tables.
MCNP data	Reviewed data files provided by other LANL teams and used by MCNP for its calculations.
NNSA	National Nuclear Security Administration, a part of the U.S. Department of Energy
Razor	A software tool from Visible Systems that combines issue tracking, version control, and release management.
Regression testing	Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements (IEEE Std. 610.12-1990).
Release Manager	The person responsible for assembling and distributing an intermediate or major release of a code baseline.
RSICC	Radiation Safety Information Computational Center in Oak Ridge, TN.
SDD	Software Design Description
SQA	Software Quality Assurance
SRS	Software Requirements Specification
Team (MCteam)	One or more members of the Eolus Project development team; this role may include other people from X-5 and other LANL groups.
Team Leader (MClead)	X-5 Monte Carlo Team Leader or appointed alternate
Test case	A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular

	program path or to verify compliance with a specific requirement (IEEE Std. 610.12-1990).
Test set	A set of test cases packaged for easy execution with expected results for comparison.
Unit testing	Testing of individual hardware or software units or groups of related units (IEEE Std. 610.12-1990).
Validation	The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements (IEEE Std. 610.12-1990). The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model (AIAA G-077-1998).
Verification	(1) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (2) Formal proof of program correctness (IEEE Std. 610.12-1990). The process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model (AIAA G-077-1998).

See Glossary in Razor Users' Manual for additional definitions of terms related to Razor.

APPENDIX B.

SOFTWARE DEVELOPMENT PROCESS

The Monte Carlo team software development process can be divided into five subprocesses. These subprocesses and their relationships are shown in Fig. B1. The Features, Enhancements, and Bugs subprocess is used to track the development and integration of a new feature, an enhancement, or a bug fix to a Monte Carlo team code. This is the primary or default subprocess of the Monte Carlo team software development process. The Releases subprocess will be used to track an Intermediate or Major Release from proposal to release for general use. The Other subprocess will be used for proposals that do not fall into the above two categories such as changes to one of these subprocesses or the SQA plan. It is a process added to those in the previous SQA plan. Because each subprocess (except Working) requires one or more reviews of the product before completion of the process, there is a separate process for tracking the progress of these review meetings. The Working subprocess is used for tracking small developments, a day to a week in length, that may be part of other subprocesses. So it can be considered a subprocess of any of the previous four subprocesses.

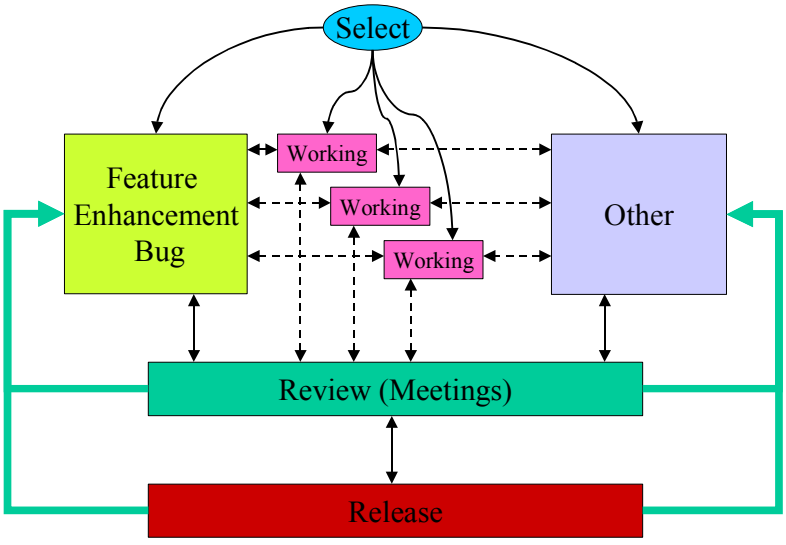

Figure B1.Monte Carlo Team Software Development Process

In the following sections, each of these subprocesses will be described in detail, one subprocess per section. Included in each section will be a narrative discussion of the steps involved, a summary of the entry conditions for each step, what is done in that step, any reviews, and the possible exits from that step. A description of the implementation of each of these subprocesses in Razor, especially in the Issues Groups, is given in Appendix C of the SQA Plan.

A detailed description of each step in each subprocess in a process description format is given in Appendix D. This description includes the dependencies, inputs, and entry conditions for starting the step, the objectives, responsibilities, related standards, and task description of the actions in the step, and the verification, exit criteria, and outputs at the end of each step.

In Appendix E, the actions required of each of the roles used in Appendices B through D are summarized. The summary for each role is divided into each subprocess in which the role appears, the state within the subprocess in which the role performs an action, and what that action is. In the process description following in this appendix, "ROLES" written in this different font refer to specific roles used in the process. These roles may have several people in them, for example a primary and a backup. The roles written in the normal font refer to specific individuals who may have that title. For example, "TEAM LEADER" refers to that role in the process while "team leader" refers to that management role.

Any of the reviews designated as a team review may be performed by the whole Monte Carlo team or a subset of it. Depending on the work product, the subset may be as small as one team member who is not the author. The documents produced as a product of many of the steps (denoted with a [D] in the subprocess figures) in these subprocesses can be produced in printed form or in electronic form. These documents, the review comments and results, and all other outputs of the subprocesses will be maintained under configuration control. These work products are summarized in Appendix F. In it, each work product is described including what resources it uses, the step in which it is created and in which it is reviewed, which steps use it, and any relevant standards.

I. FEATURES, ENHANCEMENTS, AND BUGS

The steps and process flow of the Features, Enhancements, and Bugs subprocess, to be referred to as the primary or default subprocess, are shown in Fig. B2. It has ten steps and five review meetings (three team reviews [one optional] and two Board of Directors reviews,

although the Board of Directors review that only occurs for new features is optional), making it the most elaborate of the subprocesses. It is also complicated by the fact that the results of a review meeting may return the subprocess to an earlier step or send the issue to the end of the subprocess for closeout as an outcome of a review. The return paths are not included in order to clarify the figure.

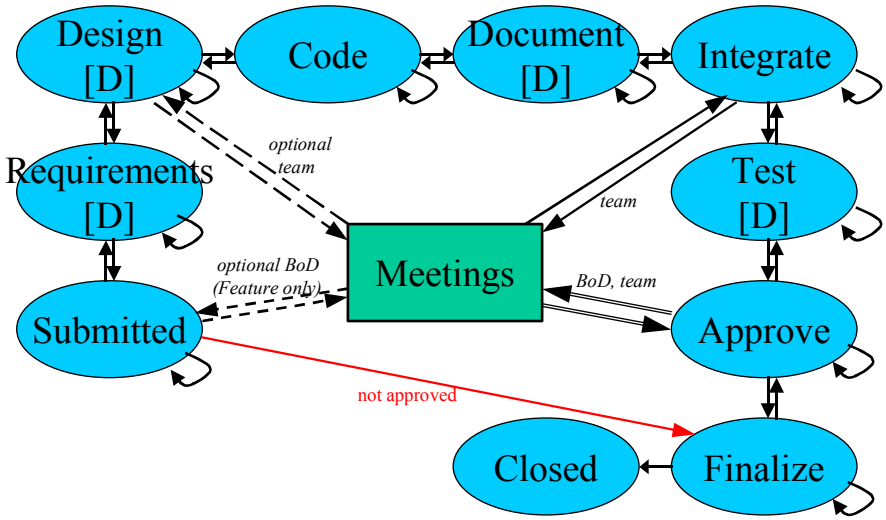

Figure B2. Features, Enhancements, and Bugs Development Subprocess

The Submitted step is the initial step of this subprocess. This occurs when someone proposes a new feature or enhancement or reports a bug. A new feature is either new or additional functionality for the code and usually requires new routines or major changes to existing routines, whereas an enhancement may be increased functionality to an existing feature, or it may be a change to the code to improve processing without changing any functionality. A bug is a minor defect in existing code and usually requires the change or addition of only a few lines of code. However, if an enhancement request or a bug report leads to major changes in the code, it can be promoted to feature status and require the additional degree of formality and review.

The proposal must include a title, a description of the issue, and a reference that may have more detailed information about the submission. The author submits the proposal when the entries are complete. The TEAM LEADER then reviews the submission. In this step, if the proposal is for a new feature, the Board of Directors may also review the proposal. However, any decision by the Board of Directors is only advisory to the team leader because the team leader has budgetary and performance responsibility for the team. If the proposal is accepted, the TEAM LEADER will assign a priority and a DEVELOPER, and it can be promoted to the Requirements step. If the proposal is deferred or is rejected, it can either remain in the Submitted step awaiting revision, or it can be sent to the Finalize step and closed out. In either case, the reason for rejection or deferral must be included before the subprocess can be moved out of the Submitted step.

In the Requirements step, the DEVELOPER determines *what* must be done to produce the proposed product or to correct the bug. The DEVELOPER takes the proposal and investigates it to determine the detailed requirements of the feature, enhancement, or bug fix and documents these requirements. For a bug, if the investigation shows that the bug is an isolated problem with no additional side effects, the documentation may be just a short clear description of the bug, a page or less, or just comments in an issue. For a new feature or an enhancement, the documentation will be more detailed. It could be a Research Note or a full Software Requirements Specification. When the requirements are documented and placed under configuration control, the DEVELOPER advances the proposal to the Design step.

During the Design step, the DEVELOPER determines *how* the product is to be created to meet the requirements and documents his or her efforts. If the product feature is complex, separate preliminary and detailed designs should be done. Any code developed during this step is considered to be prototype only and cannot be considered final.

If this design is for a simple bug fix, the design may be a simple statement of the fix, and the design description is short. Again, it can be a page or less or just comments in an issue. The DEVELOPER should include an analysis of potential side effects, if any. If the bug requires a complex fix, more thorough documentation will be required at each step. If the impact of the bug fix is severe, the reported bug should be resubmitted as a proposed feature change or enhancement.

During this process, the team may hold a review meeting to verify that the requirements and design are complete and correct and that the requirements are testable and don't have other

defects. The review of the requirements and design of a bug fix should verify that no undesirable side effects have been missed. If, during the review, the requirements are found to be incomplete, incorrect, untestable, or have other defects, the process is returned to the Submitted step, and the requirements and design are revised. If the design does not pass the review, the process remains in this step until the requirements (if necessary) and design are revised and pass review. In any case, the results of the review are documented.

Once the requirements and design have been reviewed and approved, the proposed change is promoted to the Code step, and the developer can start the actual coding of the product. Any code produced in earlier steps should be fully reworked based on the final accepted design. Unit testing by the developer to verify that the new or modified code works as designed is included in this step. The results of unit tests should be repeatable and compared with results from the regression test suite. Specific integral tests may have to be developed. Peer reviews by team members may be done at several points during the coding step. When coding and unit testing are complete, the project is promoted to the Document step, and the code, unit test set, and test results are placed under configuration management.

Now the documentation for the full code must be updated, if that has not already been done. For MCNP, changes for Chapters 2, 3, and any other affected chapters or appendices of the Users Manual must be produced. The developer has the primary responsibility to see that this is done. Only when the documentation updates have been completed can the DEVELOPER promote the activity to the Integrate step. This promotion to the next step does not mean that the documentation of the code itself is complete, only that the changes to the manual are to be provided in separate form and placed under configuration management.

When the developer has completed coding, unit testing, and documentation of this project, the team performs a line-by-line review of the code and compares it with the approved requirements and design. The team also reviews the documentation. If rework of the code or documentation is required, the developer revises it and resubmits it for review. If the developer while coding or the team during the review determines that design changes are required, the project is demoted to the Design step, and reenters the subprocess flow from that step. If the developer or the reviewers find the requirements to be incomplete or incorrect, this project must be returned to the Requirements step. After the requirements have been reworked, the subprocess continues from that step so that the design and coding can be revised to match the revised requirements. This cycle back to an earlier step may occur multiple times before the

code is approved. When the code with its requirements, design, and documentation pass the coding and documentation review, the review results are documented, and the process continues in the Integrate step.

In order to promote the project to the Integrate step, the TEAM LEADER must assign an INTEGRATOR. The INTEGRATOR takes the reviewed and documented code and integrates it into the base code. If anything more than minimal changes to the new code are required, the code is returned to the DEVELOPER for correction and retesting. If the code fails integration into the base code, perhaps due to conflicts with previously integrated modifications, the code is *returned to the DEVELOPER* for modification, and the project is demoted to the Code step. When the integration is complete, the INTEGRATOR documents the results of the integration, and promotes the project to the Test step.

After the feature, enhancement, or bug fix has been integrated into the base code, the test suite must be run in all processing modes to ensure that the full code including the new feature, enhancement, or bug fix works properly on all supported platforms. To complete this, the test suite must be modified or expanded to test the new code and perform regression testing on the rest of the code. This testing may be done by one or more members of the team. If the testing does not produce acceptable results from one or more of the tests, the process must be returned to the appropriate step (Requirements, Design, or Code) depending on the root cause of the failure. When the newly integrated code passes all tests, the test suite documentation has been updated to include the modifications for testing the new code, and these and the test results have been placed under configuration control, the project can be promoted to the Approve step.

At this step, the development of the new feature or bug fix is done. The entire development package (requirements, design, code, documentation, and test results) is reviewed by the team to complete the verification of the feature, enhancement or bug fix. For a simple bug fix, this package may be very small. Also, for new features only, the Board of Directors holds a review meeting of the package to verify that the user interface, input and output, is correct and complete. If the review or reviews are passed, the project is promoted to the Finalize step. If the package fails a review, the project is returned to the step that needs to be redone to correct the cause of the review failure. Also, the results of these reviews shall be documented.

The only remaining tasks to be performed after approval are the releasing of the code, publishing and distributing the information about the new feature or bug fix, and the filing of the relevant documentation. Once that is complete, the TEAM LEADER can promote the project to the Closed step. Once a project is in the Closed step, it cannot be modified.

II. RELEASES

A release can be one of two types. An intermediate release can be produced when one or more issues have been integrated into the main code. A new version of the code is produced when a new feature, enhancement, or bug fix is integrated into the main code. However, the new version may not produce a new release. An intermediate release is a limited distribution release available mostly to internal LANL users. It contains the updated code, a list of changes to the code, and possibly updates to the Users Manual. There will be several intermediate releases between each major release. Because this is primarily an internal release, the Board of Directors is not involved in its approval. An Intermediate Release can be considered a form of extensive user testing.

A major release has been produced approximately once every three years. This is a release to the international user community. It contains all new features and all bug fixes produced since the last major release. It includes not only the new baseline code and updated Users Manual, but also contains updated installation procedures, regression test sets, data files, and test results. It becomes the baseline for all future development.

The subprocess used for a release is shown in Fig. B3. It begins when the RELEASE MANAGER proposes a new release. The release may be prompted by the completion of a new feature(s) or by the time elapsed since the last major release. The release is promoted to the Complete step where the RELEASE MANAGER checks all code changes, both new features and bug fixes, since the last release and verifies their integration into the base code. Then the code is frozen, and the release is promoted to the Document step.

All features and bug fixes to be included in this release are then documented in a memo from the RELEASE MANAGER. In the memo, the RELEASE MANAGER must certify that the features, enhancements, and bug fixes meet software specifications and designs, that they are properly integrated, and that they have been multiply reviewed. The team now reviews this package of features, enhancements, bug fixes, and documentation. For Major Releases, the team reviews the entire Users Manual, especially Chapters 2 and 3, to verify the manual is

complete and consistent with the code being released with it. If the package passes the review, the release is promoted to the Test step. If it does not pass review, the release is returned to the Propose step so the package can be revised. In either case, the review is documented.

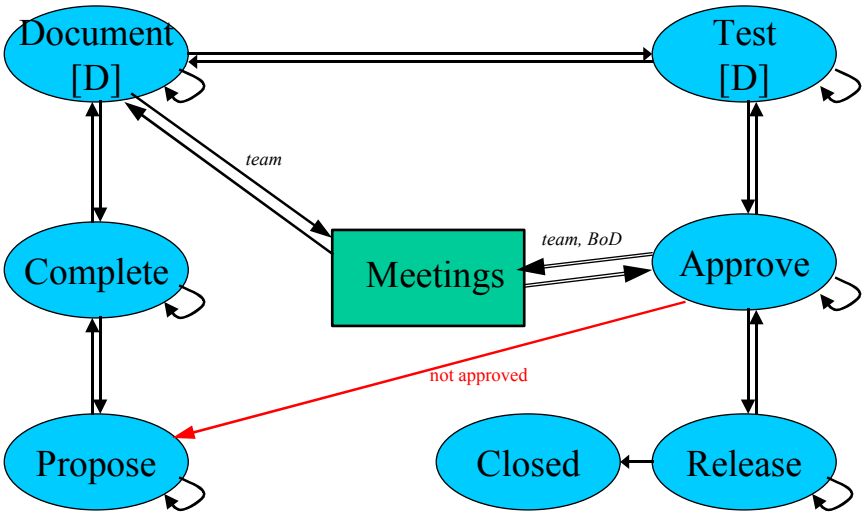

Figure B3. Software Release Process

In this step, the TESTER reviews the test results from the integration of the new features and bug fixes. If they cover completely the proposed release, then no further testing is done. However, if there are gaps in the coverage, the release is then run through the entire test suite to make sure everything works as described and that nothing has been broken while preparing this release. Also, the test suite documentation shall be updated to reflect this release, its updated test suite, and the results of the testing. When that is complete, the release is promoted to the Approve step.

For Intermediate Releases, the team now reviews documents that specify the user interface and describe the testing and validation of the features, enhancements, and bug fixes. If the Intermediate Release passes this review, it is promoted to the Release step. For Major Releases, the team must review this package, and both the team and the Board of Directors must review the user interface to verify that it is ready for international release. Once these

approvals are obtained, the release is promoted to the Release step, and the package is now ready for release to the user community. In all cases, the results of the reviews are documented.

After the code has actually been released to the internal users for an Intermediate Release or to the Radiation Safety Information Computer Center (RSICC) for a Major Release, the TEAM LEADER makes a final check to verify that everything is complete. With that verification completed, the release is promoted to the Closed step.

III. OTHER

This subprocess is used to track Monte Carlo team issues that do not involve changes in a code itself. These include such things as changes to the processes, changes to its implementation in Razor, and all other changes outside the codes. This subprocess is shown in Fig. B4 and is a simplified version of the primary subprocess. If the proposed product includes code, it will probably be between a bug fix and a new feature in size and complexity.

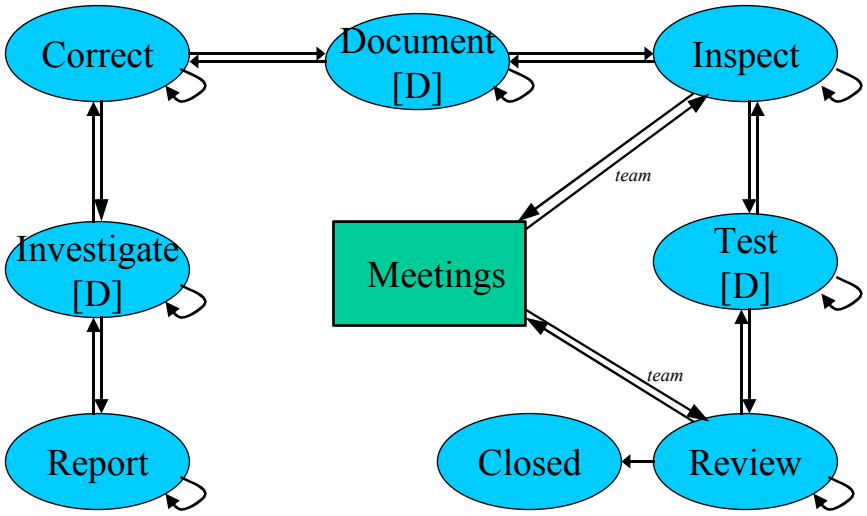

Figure B4.Other Subprocess

When a proposal for an Other item is made, it starts in the Report step. The proposal will contain the same information as a new feature, enhancement, or bug fix proposal: title, reference, and a description. The TEAM LEADER verifies that the proposal is complete, and if accepted, assigns a priority to the proposal and a DEVELOPER to investigate the details of the proposal, documents the review, and promotes the proposal to the Investigate step. If the proposal is rejected, the process is moved to the Review step and closed out.

The Investigate step is similar to the Requirements step in the primary process; namely, the developer analyzes the proposal and develops the requirements to implement it. Once investigated, the DEVELOPER will promote the proposal to the Correct step.

Here, the developer designs and creates a "correction" based on the requirements. When the "correction" is complete, the proposal is promoted to the Document step. The developer will then insure that the "correction" and the results of the investigation have been properly documented. The proposal can then be promoted to the Inspect step.

This "correction" and its documentation are then inspected by the team or by the team leader. If the "correction" fails the inspection, the process is returned to the Investigate step. Otherwise, the process is promoted to the Test step. This "correction" is now "tested." Obviously, some proposals may not be able to be tested in the conventional sense. If the proposal passes testing, it is promoted to the Review step. If it fails the testing, it is returned to the Investigate step for rework.

Now the proposal item is complete and needs to be reviewed. The team will review the entire package, the "correction," its documentation, and the testing results. If the "correction" fails this review, the proposal is returned to the Investigate step for rework. If the review is passed, the "correction" can be released, and the process closed.

IV. MEETINGS

The Meetings subprocess covers the steps related to holding a review or inspection required by one of the other subprocesses. Each review will be linked to one or more proposals in those subprocesses. Although this is the Meetings subprocess, an actual meeting to perform the review may not be held. The Meetings process is shown in Fig. B5.

When a review is required by another subprocess, the proposed review starts in the Schedule step. In this step, the Board of Directors chair or the moderator appointed by the team leader determines what items are to be reviewed, who will do the reviewing, and he or she

schedules the review. The moderator will notify the participants and provide the items to be reviewed early enough for the reviewers to have adequate time to perform their review.

Figure B5. Reviews Subprocess

When the meeting convenes, the moderator determines if the reviewers are adequately prepared for the review. If the meeting does not proceed because of lack of a quorum, or the participants are not prepared, or any other similar reason, the meeting is rescheduled, and the issue remains in the Schedule step. If the reviewers are prepared and a quorum is present, the meeting continues. If the review cannot be completed in the time allotted, it is recessed and continued at a later time. The proposal remains in the Schedule step. After the review is completed, the comments and results are documented.

Once the review and its documentation are complete, the proposal is promoted to the Actions step where the action items produced in the review are worked. After all the action items are completed, the moderator reviews the results of the action items and related documentation for completion. The moderator may decide further work is needed. He or she returns the items to the author for rework, closes this review, and returns the parent process to the appropriate previous step. The moderator may decide the completed work needs only another review. He or she then schedules and holds the additional review. When everything is complete and the moderator is satisfied, the review can be promoted to the Closed step, and the parent process can proceed.

V. WORKING

The Working subprocess covers low-level activities that are performed as part of the development of work products in another subprocess. These activities may be coding a subset of a new feature, writing part of a document, or working an action item from a review. It allows more detailed tracking of activities that may take a day to less than a week to do. This subprocess has two states: Active and Closed. The Working process is shown in Figure B6.

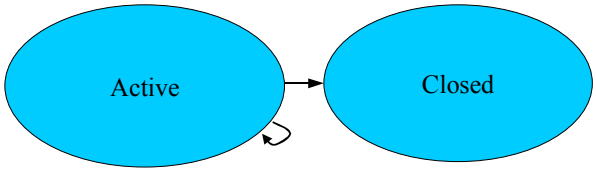

Figure B6. Working Subprocess

A new proposal is created in the Active state and requires only a title and a description of the proposed activity. The identification of the developer and related issues should be included. As activities proceed, the proposal should be updated describing the activities completed. When the activities covered by this proposal are complete, the description of the activities covered should be completed, and the proposal is moved to the Closed state.

APPENDIX C.

RAZOR IMPLEMENTATION

Razor is a software product designed as an integrated problem tracking, configuration control, and release management tool. It was developed to run in a Unix environment and has been extended to run in a Microsoft Windows NT environment on personal computers. Razor is divided into three elements: Issues, a problem tracking system; Versions, a file version control system; and Threads, a build management tool. In this appendix, each of these elements will be described. Also, information on how to access Razor will be provided.

The following outlines the use of Razor to implement the software development process described in this SQA Plan. When a new feature or enhancement is proposed or a bug is reported, it is entered as a new issue in the primary or default Issues group. This issue is then used to track the development, coding, testing, and reviews of the proposal until it is complete, and the code is ready for release to users. If the completed issue leads to a new intermediate release, the Release Issues Group is used to track that part of the process. When it is time to produce a new major release, the Release Issues Group is used to ensure that all necessary steps are completed. The Other Issues Group is used to track issues not directly part of code development. These issues include, but are not limited to, changes in the process and changes in the Razor implementation. The Meetings Issues Group is used for the various reviews required by the SQA Plan. These include features reviews, design reviews, code reviews, and release approvals. The meetings are linked to the related issues so that an issue cannot have its state changed without the required review being completed. The Working Issues Group can be used to track activities performed as a subset of a step in any of the other Issues groups.

The Versions element of Razor is the code repository and configuration management tool. The code is stored in its database burst into a separate file for each module, subroutine, or function instead of the old method of including the entire code in one file. As a subroutine or function is needed, it can be checked out of the repository by identifying the issue or issues related to the check out. The routines can be saved to a separate branch while in development, but can only be checked back into the trunk or base module after proper reviews and integration. An issue cannot be closed until all related branches have been checked back into

the trunk. The repository can save any type of file, not just code. This includes scripts controlling Razor, documentation files, and other project files. More information on this is provided in Section II.

Threads is a tool used to manage releases of a product. It is used to tie together various items in the Versions database to generate a release. It is not limited to code modules, but can also link documentation and other files. A project can be created to link several threads together. This may occur when the code modules are tracked with one or more threads while the documentation is tracked with other threads. Threads will be linked to a release in the Release Issues Group.

Roles in Razor control what individual can do what action and when he or she can do it. The roles currently defined in this implementation of Razor are RAZOR_ADMIN, SQA, BoD, team member (MCteam), team leader (MClead), Developer, Integrator, Tester, and Release Manager (RelMgr). The functions of these roles were described in the subprocesses. A summary of the functions performed by each role is given in Appendix E. Individuals may be assigned to one or more roles, and one or more roles can be assigned to other roles. Each role contains backup, and the SQA role is a member of all other roles. These assignments may change as implementation and use of this software development process and Razor proceed.

I. ISSUES

The Issues element of Razor has been configured to track proposed features and enhancements, reported bugs, proposed releases, other issues, and the reviews related to them. Each issues group requires several items of information in order to document the issue and track and control its movement through the process. Therefore, each issue has a series of states, each of which corresponds to a step in the process. In order to advance to some of these states, reviews must be held and documented in the Reviews Issues Group. If properly formatted, the text blocks in a Reviews Issues Group form can be directly used to produce the meeting agenda and minutes.

To start the Issues tool of Razor on a system that has been set up as described in Section IV below, the following characters are entered at the Unix prompt.

issues &

After a series of messages have been displayed in a startup window, a window similar to that shown in Fig. C1 will be displayed. Details about the contents of the window and the use of the icons and functions are found in the Razor Users Manual.

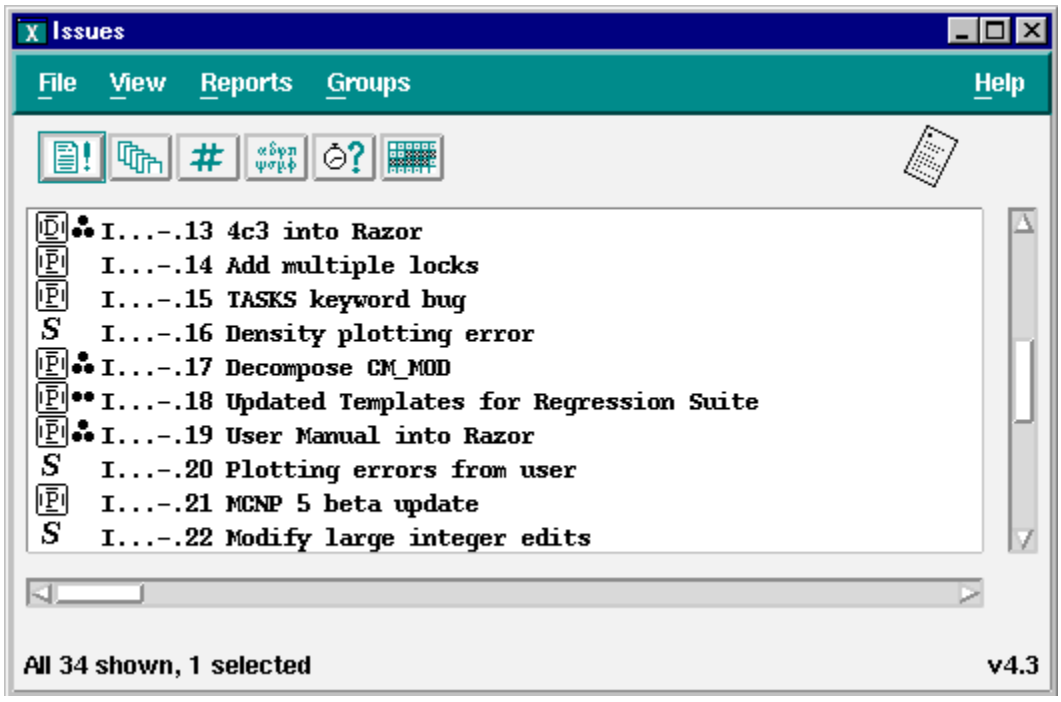

Figure C1. Issues Main Window

In the scrollable region, a list of the existing issues in the Issues group identified at the top of the window is displayed. The leftmost icon indicates the state, and the large dots are used to indicate the priority, one for the lowest and four for the highest. Each issue number (the periods are place holders) and its title are displayed. To select an issue, just click on it, and an issue form with that issue's information will be displayed. To create a new issue, either select File, and then select New Issue from its menu or click on the leftmost icon on the tool bar just below File.

To select a different Issues group, select "Groups," and an alphabetical list of Issues groups will be displayed. Click on the desired group, and the list of its issues will be displayed.

A. Issues Forms

When an issue is selected, a window with a form similar to one of those shown in Fig. C3 through Fig. C7 will be displayed. Each Issues group has its own issues form, and the groups and their associated process will be discussed in the following sections. Although the Issues forms are different, they have many things in common.

Each form has three sections. The first section contains the attributes of that issues group. The first attribute is the Title, or the Version for the Release Issues Group. The Title or Version must be filled in for the issue to be accepted. This attribute is followed by one or more attributes such as Reference, Priority, Developer, Integrator, Related Issues, or projected release date. These attributes may not have to be filled in until later, and they may have limits as to what roles can enter or change them. Below the attributes is a large block in which the states for the issue are listed. As the state of an issue is changed, including to the same state, the date, the time, and the username of the person who requested the change are displayed on the line of the new state.

The next section is the first of two text blocks and is the Problems or Description section. A description of the feature, bug, meeting agenda, or release must be entered. It need not be detailed because details can be provided in the Reference. However, it must contain enough information so that the issue is understood.The last section on the form is the second text field and is the Actions Taken section. Every time anything is changed on the form, an entry must be made therein. A specific format has not been defined yet. However, the requestor should put his or her name or initials and a date and time at the beginning of the entry. This field can provide a history of actions taken on an issue, and in the case of the Meetings Issues Group, be used as the minutes of the meeting.

Razor automatically creates the issues form window at what it determines is the height of the screen. This may be larger that the usable area of the screen. This window can be resized using common methods. The sections inside the window can be resized by placing the cursor in the box toward the right-hand side of the separator line and moving it up or down as necessary. The issues forms shown in the following figures are sized to show the attributes. As a result, the text blocks at the bottom of the issues form are much smaller than normally displayed.

Each field on the Issues form will be checked to see if it is in the proper state or not. If an invalid entry is found in a field, a window such as that in Fig. C2 will be displayed. If the item is not in the proper state, the message displayed will indicate what must be changed. Explanations of the messages produced by the scripts and what must be corrected are given in Appendix C of Ref. 5. Pressing the button will return input to the issues form to correct the error.

Figure C2. Error Message Window

B. Default Issues Group

The issues form for the default issues group is shown in Fig. C3. In this and the other figures of Issues forms, the Feature Description and Actions Taken text boxes have been minimized to expand the size of the attributes box display. The form begins with a Title text line, a Reference text line, and a Type selection. These must be completed for the form to be accepted. The Feature Approval and Priority are set by the TEAM LEADER after he or she reviews the proposed feature. If the feature is approved, the issue can be promoted to the Requirements state. Otherwise, the issue can only be promoted from the Submitted state to the Release state.

The process associated with this form is described in Appendix B, Section I. The promotion of an issue from the Submitted state can only occur after several requirements have been met. In addition to approving the proposal and setting the priority, the MCLEAD must also assign the DEVELOPER. A BoD review of a proposed new feature may also be done. Reviews are also required to promote from the Design state to the Code state and from Integrate state to the Test state. Only the DEVELOPER can promote an issue from the Requirements state and from the Code state. Before the issue can be changed to the Integrate state, MCLEAD must assign an INTEGRATOR, and only the INTEGRATOR can change the issue from the Integrate state. Only someone in the TESTER role can move the issue out of the Test state.

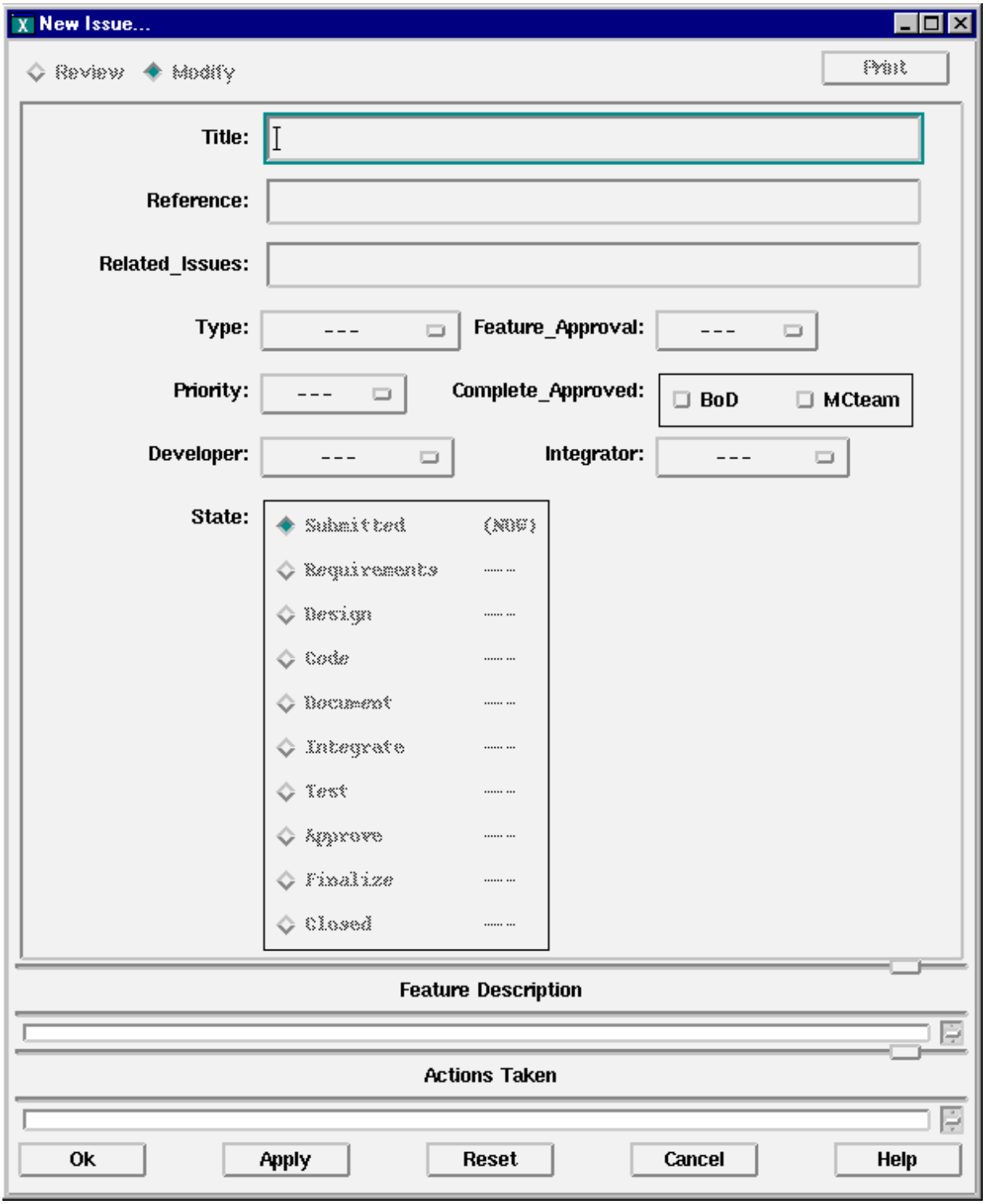

Figure C3.Features, Enhancements, and Bugs Issues Form

Before the issue can be promoted to the Release state, both the BOD and MCTEAM must complete their reviews and mark the appropriate Complete Approved box. Only the MCLEAD can close an issue. Most other state changes can be done by anyone in MCTEAM. Note that an issue can be demoted one or more steps if it fails a review required for a promotion from a state. Details of each step in this subprocess are found in Appendix D, the actions required of each role are found in Appendix E, and the work products from each state are described in Appendix F.

C. Release Issues Group

Figure C4 shows the Release Issues Form. The only attributes this issues group has in addition to its states are the Version, Release type, projected release date and release approval. The release approval applies only to a Major Release. This issue group has two reviews required. One is a documentation review, and the other is the release review. The Board of Directors is only involved in the Major Release approval review. The process associated with this form is described in Appendix B, Section II.

D. Other Issues Group

This issues group is the catchall issues group, and its form is shown in Fig. C5. It can be used to track modifications in the process, it can be used to track changes in this implementation, and it can be used to track any other issue that does not fall in the default Issues Group or the Release Issues Group. These issues are usually team internal issues. Although it is general in application, there are two possible reviews. One may be to review "correction" and documents created during the process. After the Document state is complete, the products of this issue are inspected by the team. Also, the products may be tested, and the team will review the results of any tests. However, there are fewer other restrictions as to who can do what. Otherwise, it is very similar to the Features and Bugs Issues Group.

E. Meetings Issues Group

The Meetings Issue Group is different from the previous issues groups because it is a child group and the three previously described groups are parent groups. The Meetings Issue Group form is shown in Fig. C6. Some of the fields are the same as those in other issues groups, especially the Release Issues Group. However, this form has attributes fields that none of the other issues groups have, namely, Location and Issues. The location is self-explanatory.

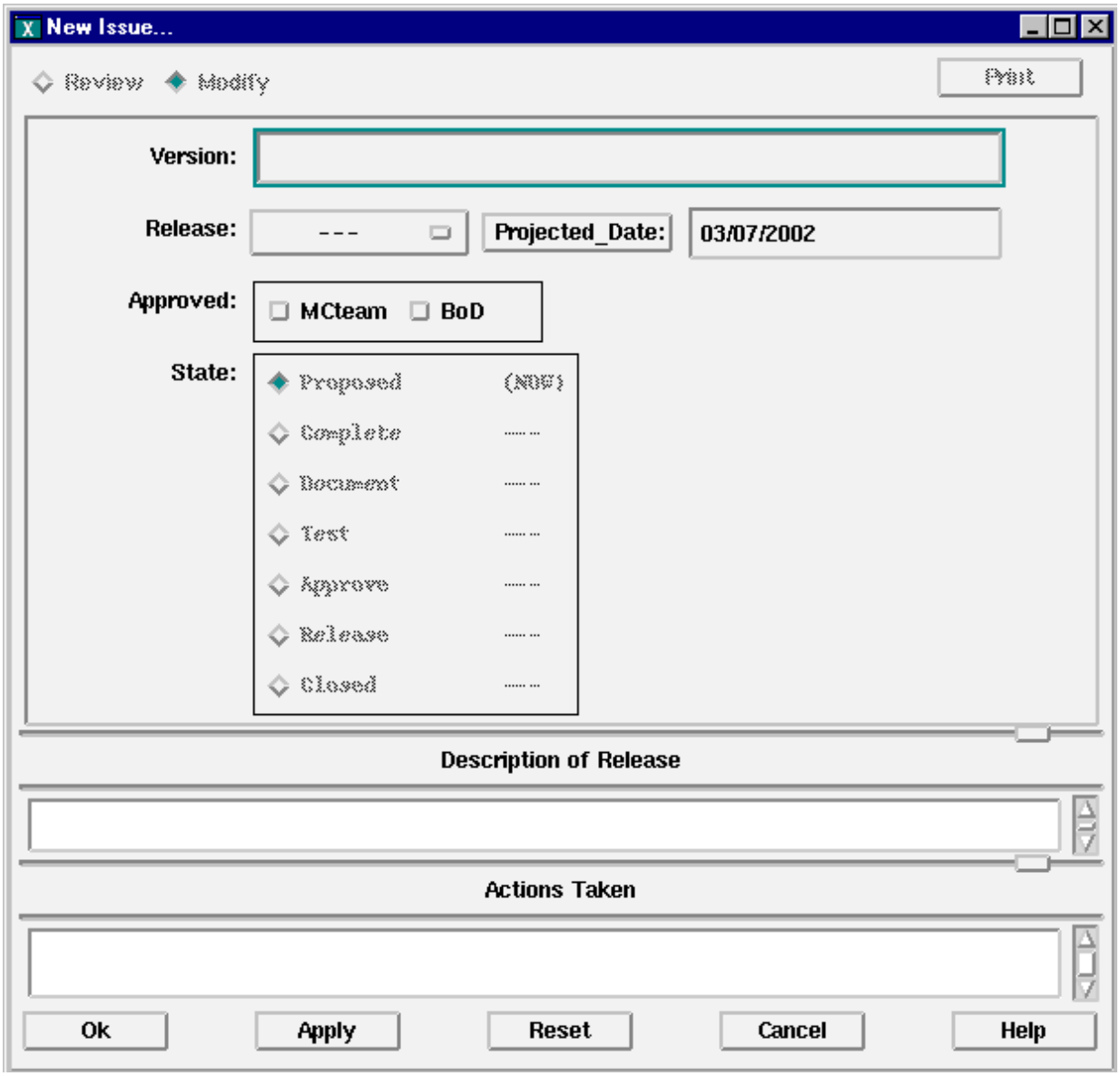

Figure C4. Release Issues Form

The issues entered in the Issues field are the parent issues that are then linked to this child issue. This allows the parent issues to be able to check on the status of the child issues as part of their state promotion process.

The reviews reported in this issue group do not have to be big elaborate, formal meetings. They can be simple two or three person meetings in an office to perform the review. All that is required is that the results are documented in this issues group. The agenda is a description of

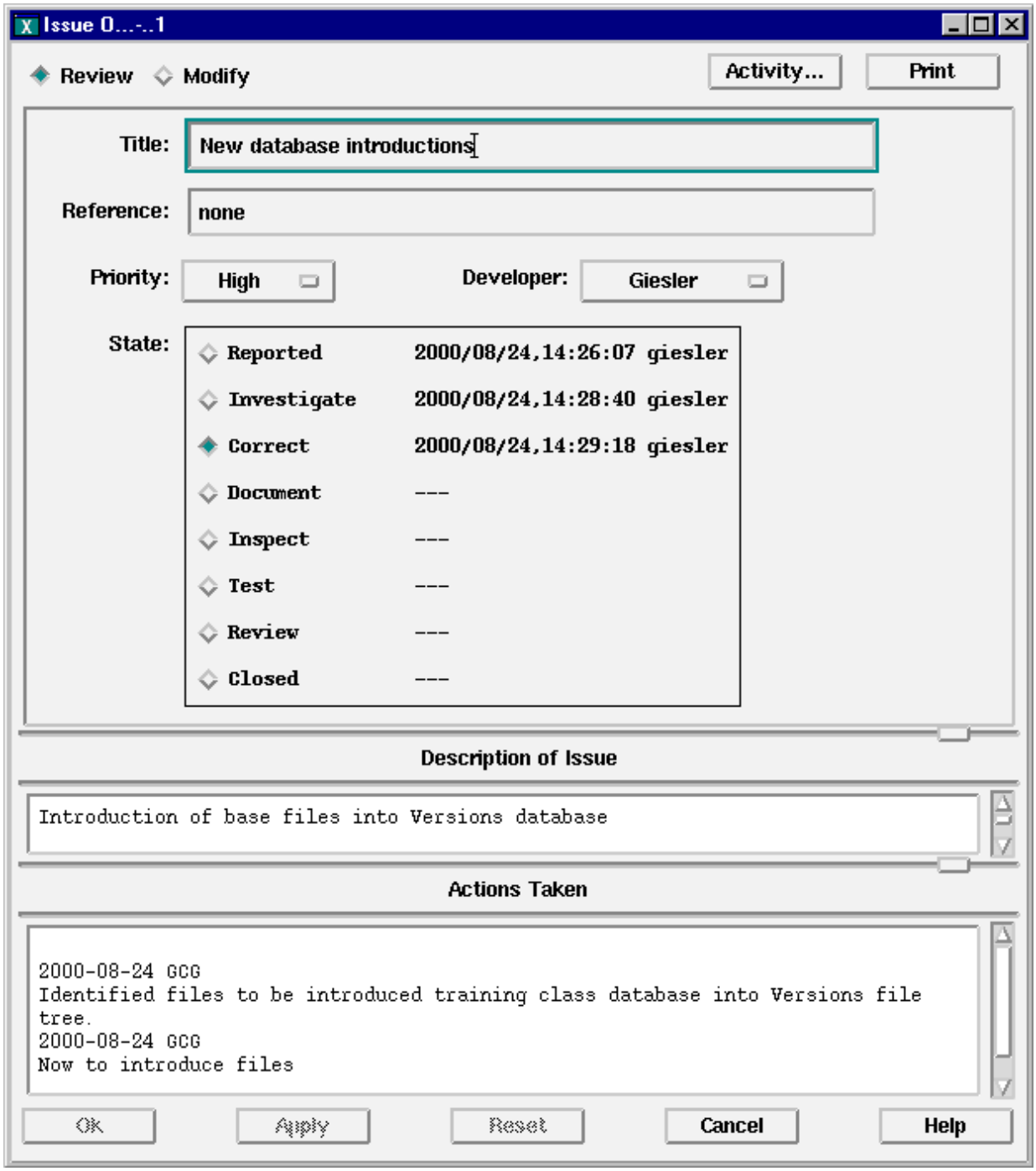

Figure C5. Other Issues Form

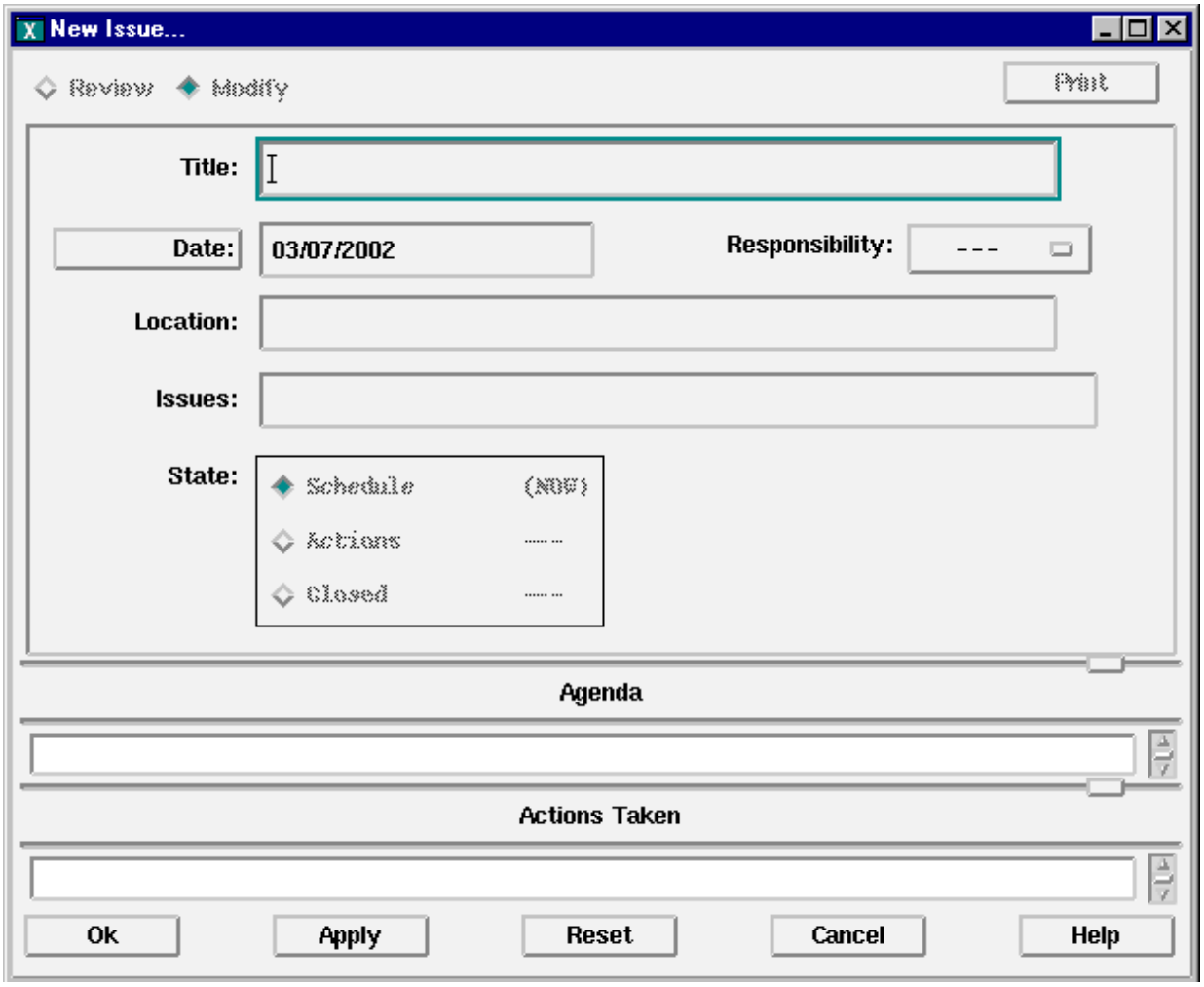

Figure C6. Reviews Issues Form

The issue or issues to be discussed in this review and may be generated from the list in the Issues text line. The entries in Actions Taken can be extracted and used for the minutes of the review. The process associated with this form is described in Appendix B, Section IV.

F. Working Issues Group

The Working Issues Group is similar to the Reviews Issues Group in that it is a child issues group. However, it can be a child of any other issues group including itself. Its form is shown in Figure C7. This form has only the Title, Developer, Issues, and State attributes. Although it has a Issues attribute like the meetings issue form, it is currently not activated as a required link to the parent issue.

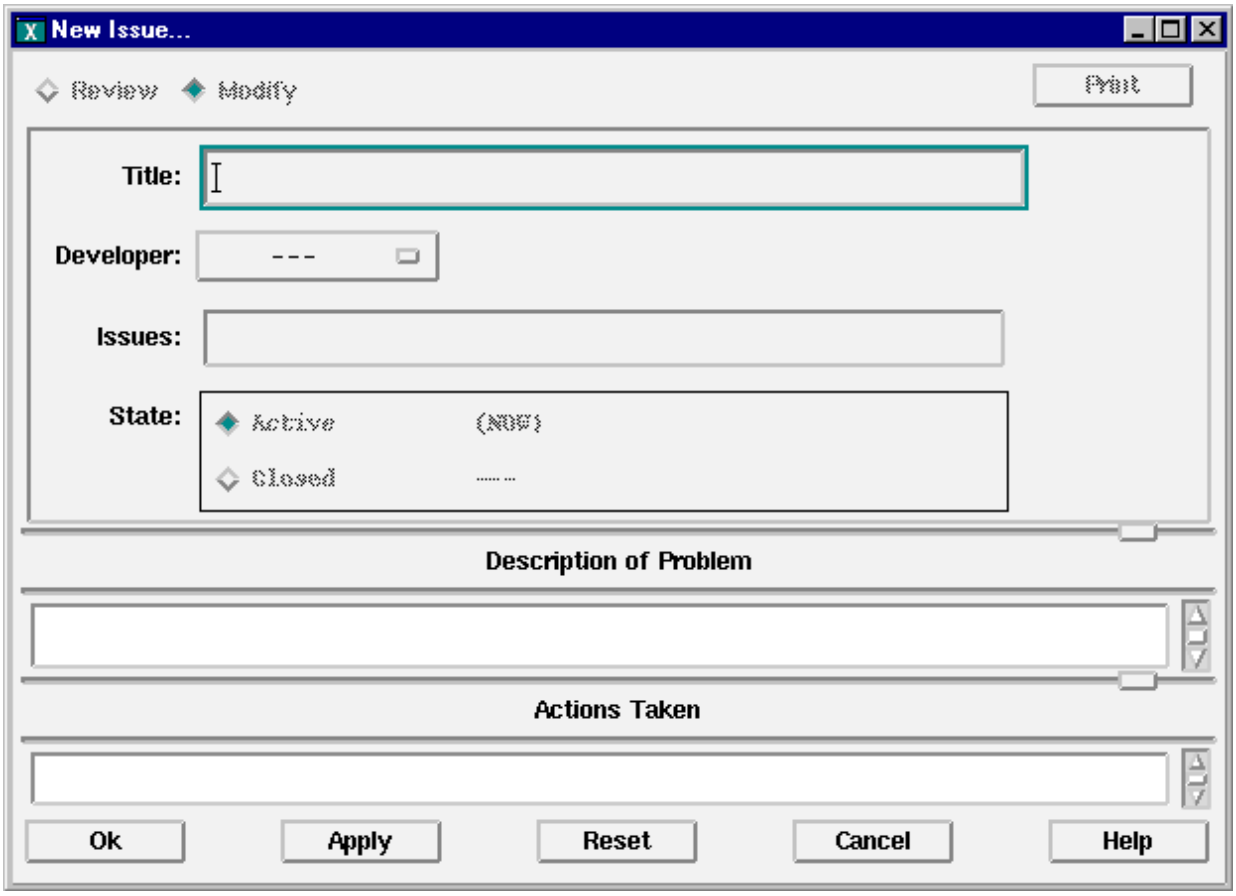

Figure C7. Working Issues Form

II. VERSIONS

Versions is the configuration management system of Razor. It can contain the source code and any other related files such as scripts and documentation. It allows tracking and controlling access and changes to any of the files. The main development path of a file is known as the trunk. Developers can branch a file and the check out the branch for use in development without affecting the trunk copy or other users. The trunk is locked against updating by anyone other than the INTEGRATOR.

Versions is started by entering the following at the Unix prompt.

versions &

In the IRIX environment, the following must be used to start Versions because IRIX defines an internal tool to respond to the versions command.

$RAZOR_HOME/bin/versions &

After a series of messages have been displayed in a startup window, a window similar to that in Fig. C8 will be displayed.

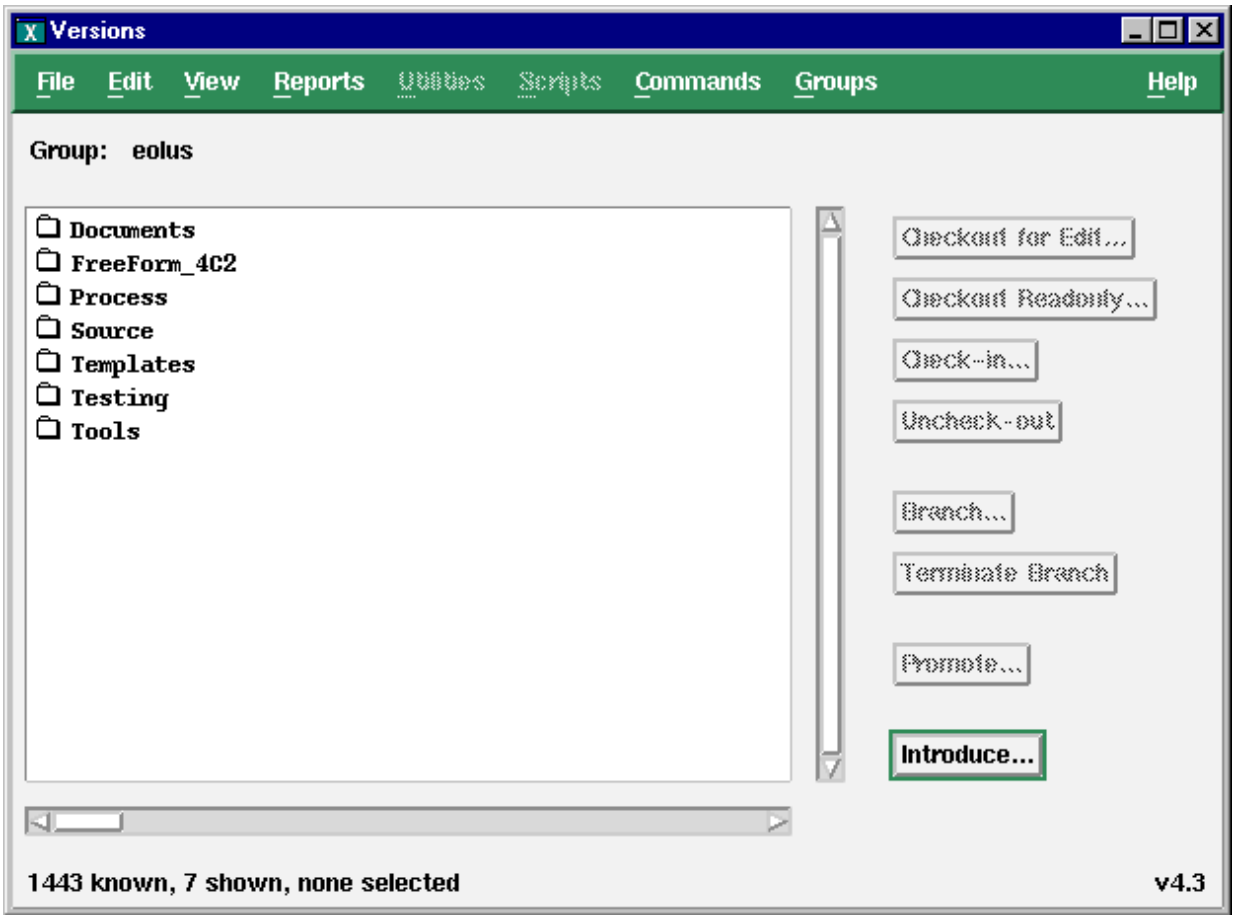

Figure C8. Versions Main Window

Whether or not some of the buttons on the right-hand side of the screen are active depends on the item selected and its current state. If a folder is selected or nothing is selected, a new file can be introduced or the folder can be opened to display a list of files in that folder. If a file is selected, it can be checked out for edit or as read only, or a branch can be created to use for development. If the trunk is locked, which should be the normal case, the file can only be checked out from a branch.

The current Versions folder tree includes Documents, Process, Source, Templates, Testing, and Tools. Each folder will have a number of folders below it containing specific categories of items. The Documents folder will contain copies of requirements specifications,

design documents, users manuals, Research Notes, and all other project related documents. The Process folder will contain the scripts used in this process and standards related to this process. The source code itself will be found in the Source folder. Code module templates and document templates will be found in the Templates folder. The Testing folder will contain both the test decks and copies of the testing results from both regression testing and unit testing of new features. Test results templates will also be found here. Finally, the Tools folder will contain the executables of various tools that can be used in code development.

Figure C9 shows a copy of the window used to create a branch of a file. As usual, a title is required. The description is optional. A list of all the available versions of the file is displayed for selection of the desired version. One or more issues must be associated with the branch being created. Directions on how to do this are in the Razor Users Manual Chapter 4. When the branch is created, it will be listed in the Versions window immediately following the parent file. Figure C10 shows the results from branching several files. Note the "…" for file name and the appended branch numbers to the version number. This branched file is a copy of the trunk file, but it can be checked out for editing. The DEVELOPER will have the option regarding where to store it. By default, it will be stored in a folder tree off the home directory that is the same as the Versions tree. In this example, a branch of Depends will be stored in /home/*user*/Source/src/Depends. It can then be processed as a normal text file.

Note that the list of versions in Fig. C9 contains only those versions available at the time. As development continues, new versions will be created by others and may be integrated into the trunk. Before presenting files for integration, the DEVELOPER must make sure that all branches are based on the current trunk version of each file otherwise the INTEGRATOR will reject it when presented for integration.

When coding is completed, the edited version of the file is checked back in to the branch. It may also be checked in periodically during development as an archiving measure. After coding is completed and the issue is ready for integration, the INTEGRATOR merges the

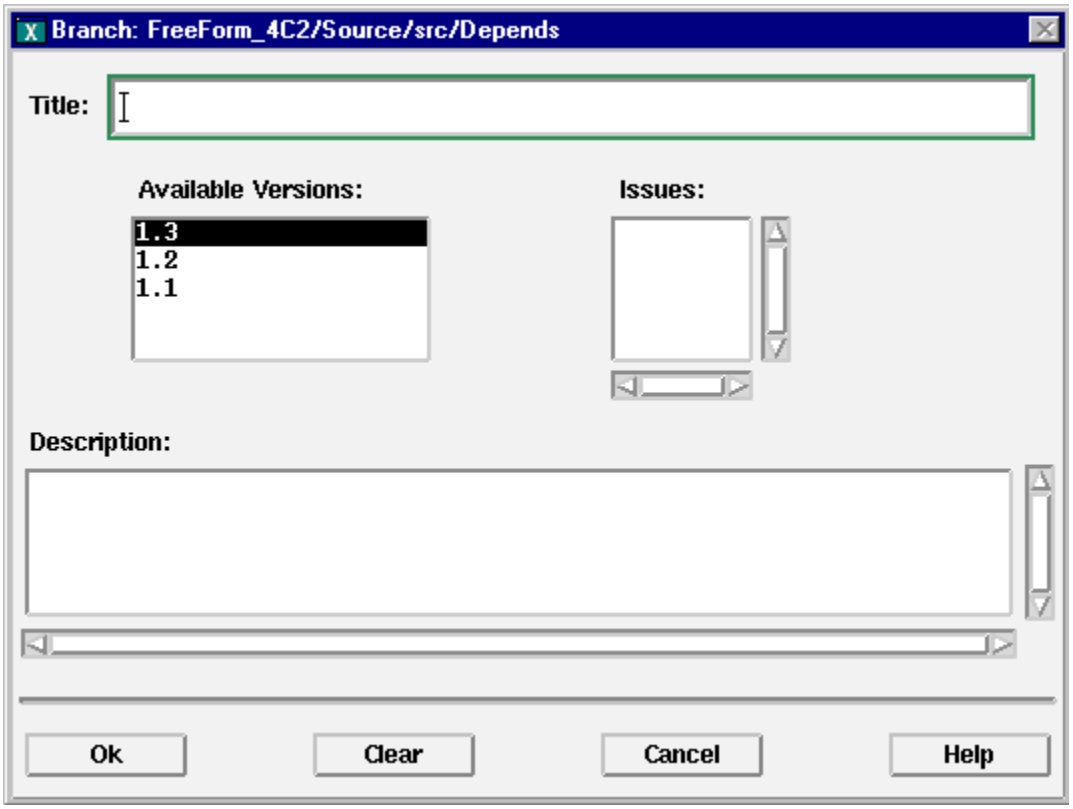

Figure C9.Versions Branching Window

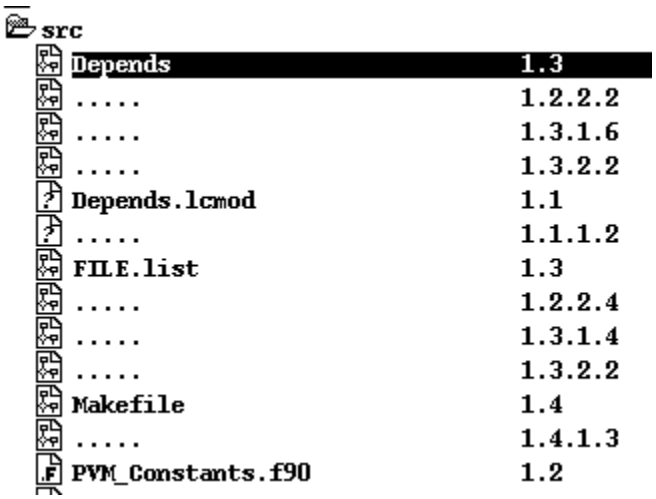

Figure C10.Window with Branched File

branch into the trunk creating a new version of that module. Dead branches can be terminated by the INTEGRATOR or SQA.

III. THREADS

Threads is the release-control tool of Razor. With it, a set of files can be defined and assembled. The releases may be an Intermediate or Major Release or the thread may be a special build used for development or testing.

Threads is started by entering the following chatacters at the Unix prompt.

threads &

After a series of messages have been displayed in a startup window, a window similar to that in Fig. C11 will be displayed. Within the window, a list of the threads that have been defined will be displayed.

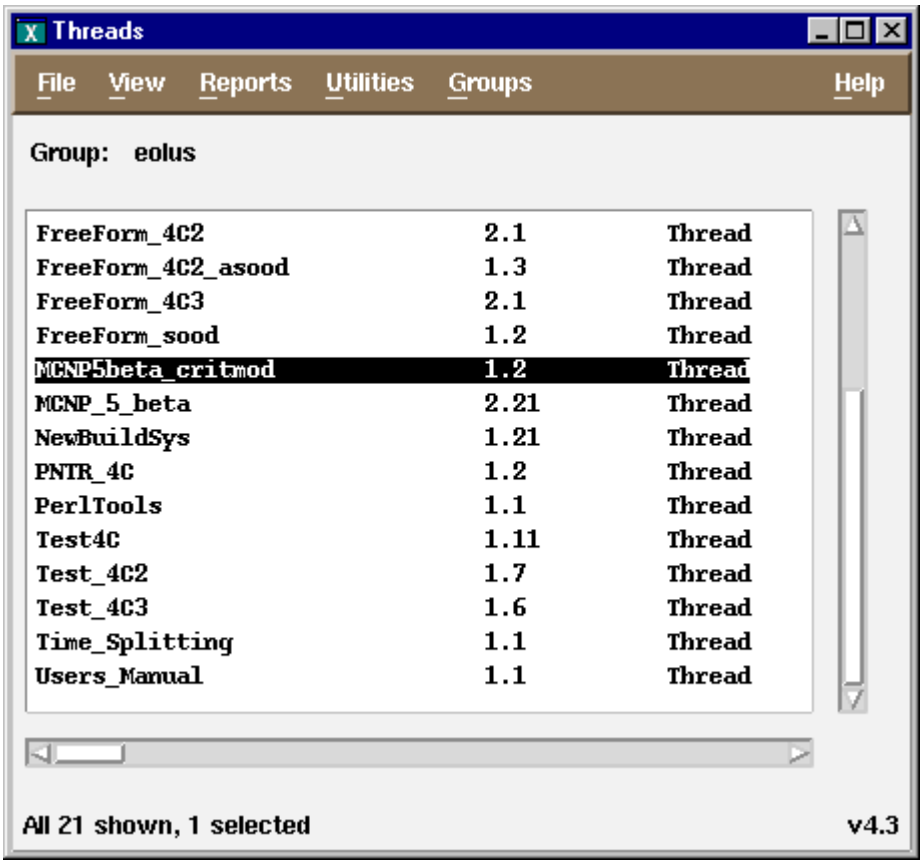

Figure. C11.Threads Main Window

In Fig. C12, a thread has been selected, and the Source/src folder within it has been opened to show the list of files. The plus sign before a file name indicates that the file has been selected for the thread. If it is desired to create a new thread, a window would be displayed

with the same list, and the user would then be able to select which files and versions to include or exclude from the list. Currently, there are no restrictions about who may create or modify a thread.

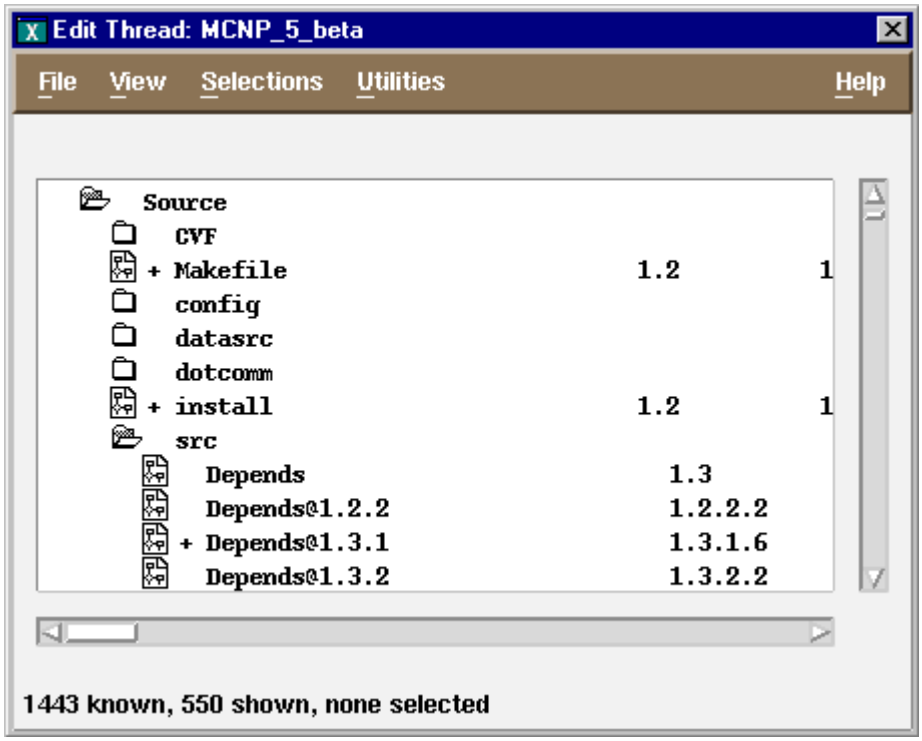

Figure C12. A Thread Window

IV. SETUP

The environment in which development will be done is located on theta. It will contain the current version of the source code, and its scripts will be configuration controlled. Changes to scripts will be made only after the issue has been processed and closed.

Setting up a user to start Razor is an easy process. All that has to be done is include one line in the login startup file. This line executes a script that defines three environmental variables that Razor uses for path definitions. This line is dependent on the Razor environment to be used. For C shell users, the line to be added to the startup file is as follows.

source /usr/projects/mcnp/dev/razor_db/rz_prep

For Bourne and Korn shell users, the line to be added to your startup file is the following:

. /usr/projects/mcnp/dev/razor_db/rz_prep.sh

If the following line is entered at the command prompt, a listing similar to that in Fig. C13 is displayed.

razor info

```
The license manager is running on thetaserver, pid = 1171

Remote Clients are using Razor Password file

Connections:
------------

Database '/usr/projects/mcnp/dev/razor_db/RAZOR_UNIVERSE',
is active on theta, rz_server pid = 3056445

----

Database '/usr/projects/mcnp/dev/lahet/RAZOR_UNIVERSE',
is active on theta, rz_server pid = 3018807

----
There are no application connections to the license
manager.
```

Figure C13.Razor Information Response

This listing tells that the license manager is running, that Razor is up serving the listed databases, and that no one is currently using any of the Razor tools. If anyone were using any of the tools, the user, the tool, and its PID would be listed. If a particular database is down, it would not be listed. If this message is not displayed at all or a warning message is displayed, the Razor license manager is down. In either case, contact the Razor Administrator to have the situation corrected.

APPENDIX D.

DETAILED PROCESS DESCRIPTIONS

I. FEATURES & BUGS PROCESS

Step 1: Submitted

Objectives	To submit a new feature or enhancement request or bug report.
Dependencies	None
Responsibilities	Any user or team member may submit an issue. Team leader reviews and approves or rejects. Team leader assigns developer and priority to approved submissions. BoD may review proposed feature paying special attention to possible user interface modifications or extensions.
Inputs	Title, reference (if any), designate as feature or bug, description of feature, enhancement, or bug.
Entrance Criteria	None
Task Description	User supplies input information to define new feature, enhancement, or bug. Team leader reviews submission, determines if it needs to be worked, and sets priority. A BoD meeting may be held to review proposed feature and reject or accept it. Multiple features may be reviewed in a single meeting. The review(s) and its(their) results are documented and placed under configuration control. If a submitted item is rejected, the process is moved to Step 9: Finalize to be documented and closed out. Otherwise, the team leader assigns a developer.
Verification	Team leader reviews decision and may modify it based on personnel and budgetary constraints.
Exit Criteria	Review and decision documented. If approved, priority set and developer assigned.
Outputs	Acceptance or rejection of proposed issue, documented decision, priority set, designer assigned to approved submissions.
Standards	IEEE 1028-1997 Standard for Software Reviews.

Step 2: Requirements

Objectives	To develop a Software Requirements Specification (SRS) for the approved feature or bug fix submission.
Dependencies	This step is dependent on Step 1 of this process including developer assigned.
Responsibilities	The assigned developer is responsible for developing the SRS.
Inputs	Approved submission from Step 1 of this process.
Entrance Criteria	Successful completion of Step 1 of this process.
Task Description	The developer determines what the product is to do based on the approved submission. The detailed requirements are determined from the raw requirements in the submission, the constraints and influences on the system, and feedback from the technical community. The analysis should include the functionality, external interfaces, performance, attributes, design constraints, and risks associated with the requirements. Each requirement should be complete, unambiguous, correct, consistent, verifiable, modifiable, traceable, and ranked for importance. The SRS can be documented in printed or electronic form. The SRS shall be placed under configuration control. If the bug is an isolated problem with no additional side effects, these requirements may be met with just a short clear description of the bug.
Verification	The SRS is reviewed and accepted by the team at the design review.
Exit Criteria	Completed SRS
Outputs	Completed SRS
Standards	IEEE 830-1998 Recommended Practice for Software Requirements Specifications.

Step 3: Design

Objectives	To create a software design description (SDD) based on the SRS.
Dependencies	This step is dependent on Step 2 of this process.
Responsibilities	The developer shall create the design. The team shall review and approve the SDD and the related SRS.
Inputs	Completed SRS
Entrance Criteria	Successful completion of Step 2 of this process.

Task Description	The developer determines how the product is to be designed to meet the requirements in the SRS. This design shall include the module descriptions, their dependencies on other modules and data, their interfaces to other modules and the process, and their detailed design. The design shall be described in an SDD that may be produced in printed or electronic form. The SDD will be placed under configuration control. If the bug is an isolated problem with no additional side effects, these requirements may be met with just a short description of the correction required. The completed SDD and SRS are reviewed and approved by the team before coding begins. Any coding done to explore or create SRS and SDD must be considered as prototypes. If the bug is an isolated problem with no additional side effects, the team leader may approve the SRS and SDD. If the SRS is not approved, the process returns to Step 2 for revision of the SRS. If the SRS is approved, but the SDD is not, the process remains in this step until the SDD is approved.
Verification	Team approval of SRS and SDD.
Exit Criteria	Approved SRS and SDD.
Outputs	Approved SRS and SDD.
Standards	IEEE 1016-1998 Recommended Practice for Software Design Descriptions, IEEE 1028-1997 Standard for Software Reviews.

Step 4: Code

Objectives	To produce the code product based on the approved SDD.
Dependencies	This step is dependent on Step 3 of this process.
Responsibilities	The developer shall develop and test the code.
Inputs	Approved SDD from Step 3 of this process.
Entrance Criteria	Successful completion of Step 3 of this process.
Task Description	The developer creates the code for the feature or bug fix based on the SDD. Any code prototyped during SRS or SDD development must be recreated to insure its suitability and compliance with the SRS and SDD. The developer also creates a test set and performs unit testing on the code to insure that the base code works correctly with the new code added, both without using the new functionality and with using it. The testing shall also determine that the new code meets the requirements. Regression testing on the modified code may be performed, if appropriate. The code may be inspected by the team. The code and test

	set shall be placed under configuration control, and the unit test set shall be passed to system test for assistance in upgrading the base code test set.
Verification	Verification shall be done during Step 6: Integration, if it is not done during this step.
Exit Criteria	Coding complete and unit tested by developer.
Outputs	Code and unit test set.
Standards	IEEE 1008-1987 (R1993) Standard for Software Unit Test, IEEE 1028-1997 Standard for Software Reviews.

Step 5: Document

Objectives	To update the Users Manual and to provide test set documentation.
Dependencies	This step is dependent on Step 4 of this process.
Responsibilities	The developer shall create or revise the documentation.
Inputs	Code and test set from Step 4 of this process.
Entrance Criteria	Successful completion of Step 4 of this process.
Task Description	The developer provides updates to the descriptions in the Users Manual Chapter 2, Geometry, Data, Physics, and Mathematics, and Chapter 3, Description of Input. If this new feature of bug fix produces changes that affect other chapters in the manual, changes to those shall also be provided. Additionally, documentation for the unit test set shall be provided. All documentation shall be placed under configuration control.
Verification	The documentation will be validated in Step 6: Integration.
Exit Criteria	Completion of documentation of the code and test set by developer.
Outputs	Code and test set documentation.
Standards	Coding Standards,[13] IEEE 1063-2000 Standards for Software User Documentation.

Step 6: Integrate

Objectives	To integrate new feature code or bug fix into base code.
Dependencies	This step is dependent on Step 5 of this process and on the team leader assigning an integrator.

Responsibilities	The team reviews the code and documentation from Steps 4 and 5 of this process. After the review is passed, the integrator integrates the new code into the base code.
Inputs	SRS from Step 2, SDD from Step 3, code from Step 4, and documentation from Step 5, all of this process.
Entrance Criteria	Successful completion of Step 5 of this process and integrator assigned.
Task Description	The team does a line-by-line review of the code, comparing it with the approved SDD. They also review the documentation to verify that it reflects the requirements in the SRS, the design in the SDD, and the actual code and that it is ready for inclusion in the Users Manual. They review the documentation on the unit test set to verify that it correctly tests the new code. They will use the unit test results to verify the correct execution of the code. If any part of the package fails this review, the process returns to the step where the failed item as created for correction of the failing item. The process will then continue from that point. The results of the review shall be documented and placed under configuration control. After all aspects of the development have passed the review, the integrator merges the changes into the base code. If there is a failure during this integration, the process is returned to the appropriate step to correct the cause of the failure and then resumes from that step. The results of the integration shall be placed under configuration management.
Verification	Successful review by the team and integration into the base code.
Exit Criteria	New code successfully integrated into the base code.
Outputs	Reviewed documentation and integrated code, documentation of review and integration.
Standards	IEEE 1028-1997 Standard for Software Reviews.

Step 7: Test

Objectives	To perform regression testing on integrated code.
Dependencies	This step depends on Step 6 of this process.
Responsibilities	Team members assigned by team leader shall perform the tests.
Inputs	Integrated code and reviewed documentation from Step 6 of this process.
Entrance Criteria	Successful completion of Step 6 of this process.

Task Description	The tester(s) shall modify the system test set to include testing of the new feature or bug fix and shall update the test set documentation. The units tests from the developer may be used to aid in this effort. The testing shall verify that the competed feature or bug fix meets all of the requirements in the SRS. The updated test set and documentation may be reviewed by the team for completeness and possible errors. The entire test set is run on all supported platforms. The results are reviewed by the tester(s) to insure the integrated code still gives the same results or explained differences for all problems and platforms. The testing documentation and results shall be placed under configuration control.
Verification	The results of this step shall be verified in the next step.
Exit Criteria	Successful test set execution on all supported platforms.
Outputs	Test set extensions documentation and results.
Standards	IEEE 829-1998 Standard for Software Test Documentation.

Step 8: Approve

Objectives	To verify and validate the results of this process for a particular submission.
Dependencies	This step depends on Step 7 of this process.
Responsibilities	The team shall review the results from integration and testing. The BoD shall review any changes to the user interface.
Inputs	Documentation from Step 5, integrated code from Step 6, and test documentation and results from Step 7, all of this process.
Entrance Criteria	Successful completion of Step 7 of this process.
Task Description	The team shall review the entire package from submission through system testing for completeness and correctness. The team shall also review the test documentation and results to insure the changes have been properly and completely tested. The BoD shall review any changes to the user interface, both input and output, which resulted from this new feature. If the package fails either of these reviews, the process returns to the step where the failing item was created for correction of that item. The process will then continue from that point. The results of these reviews shall be documented and placed under configuration control.
Verification	The team leader shall verify the completion of this step.

Exit Criteria	Approvals of the package by team and the BoD.
Outputs	Documentation of the reviews.
Standards	IEEE 1028-1997 Standards for Software Reviews, IEEE 1012-1998 Standard for Software Verification and Validation.

Step 9: Finalize

Objectives	To complete the processing the modified product for the LANL users and for consideration of the feature for an Intermediate or Major Release.
Dependencies	This step depends on Step 8 or Step 1 (for rejected submissions) of this process.
Responsibilities	The team leader shall oversee the release of the approved package.
Inputs	Approved integrated code and documentation.
Entrance Criteria	Successful completion of Step 8 of this process.
Task Description	The team leader shall oversee the release of the package to users. For a new feature, this would include the team and the local user community. This may be done through the creation of an intermediate release. A notice describing this new feature shall be sent to the team and all local users. For a bug fix, notification shall be sent to the team and local users. Announcement of the fix and its availability to all users shall be made on the web site. For a rejected submission, the team leader shall verify that a notice documenting this rejection is distributed to the team, the BoD (for features only), and the submitter. The team leader shall also verify that all products of this process are properly documented and under configuration control.
Verification	The team leader shall verify that the process is complete.
Exit Criteria	Release of integrated feature or bug fix and its documentation, and all process products are under configuration control.
Outputs	Released code and documentation.
Standards	None

Step 10: Closed

Objectives	To close the process on the submitted feature or bug.
Dependencies	This step depend on Step 9 of this process.
Responsibilities	The team leader is responsible for closing the process.
Inputs	The complete package for a released item.
Entrance Criteria	Successful completion of Step 9 of this process.
Task Description	Once the team leader has verified that the product is released, the notifications have been sent, and all items are under configuration control, this process may be closed.
Verification	None
Exit Criteria	Release complete and products under configuration control.
Outputs	None
Standards	None

II. RELEASE PROCESS

Step 1: Proposed

Objectives	To propose an Intermediate or Major Release.
Dependencies	This depends on all related features and bug fixes being integrated.
Responsibilities	The Release Manager is responsible for the proposal.
Inputs	Base code with modifications not included in the last Intermediate or Major Release.
Entrance Criteria	Proposed inclusions are integrated.
Task Description	The Release Manager determines that an Intermediate Release is needed for the recent feature(s) added or it is the time for another Major Release. He or she proposes the release with projected release date and features and bugs to be included.
Verification	The team leader concurs.
Exit Criteria	A release is proposed.

Outputs	Proposed release version and projected date.
Standards	None

Step 2: Complete

Objectives	To determine that all the included features and bug fixes are complete.
Dependencies	This step depends on Step 1 of this process.
Responsibilities	The Release Manager checks for item completion.
Inputs	List of features and bug fixes to be included.
Entrance Criteria	Successful completion of Step 1 of this process.
Task Description	The Release Manager verifies the status of each feature or bug fix to be included in the release. If any of them do not meet the requirements for being released, this process must wait in this step until all items meet requirements or the incomplete item is removed from the release. Once all items are closed, the base code is frozen against further changes until after the release is complete.
Verification	The team leader shall concur with the list of included items.
Exit Criteria	All proposed inclusions are closed, and the code is frozen. The Release Manager or team leader documents that all features and bug fixes to be included in the release meet SRS and SDD requirements, that they are properly integrated, and have been multiply reviewed.
Outputs	Documentation of all items to be included in the release.
Standards	None

Step 3: Document

Objectives	To verify that the documentation for all included features and bug fixes have been included in the Users Manual and other relevant documentation.
Dependencies	This step depends on Step 2 of this process.
Responsibilities	The team reviews the package for each item to be included in the release.
Inputs	The code and documentation package for each item to be included in the release.
Entrance Criteria	Successful completion of Step 2 of this process.

Task Description	The team members reviews the package for each item to be included in the release. In particular, for a Major Release, the team members review the entire Users Manual, especially Chapters 2 and 3. They review the manual to insure that the manual is complete and consistent with the code being released with it. For an Intermediate Release, only change pages are necessary to be included. If the documentation does not pass this review, the process is returned to Step 1 to await completion of the documentation, and then continues from there. The reviewed and approved documentation shall be placed under configuration control.
Verification	The team leader verifies that this documentation is reviewed.
Exit Criteria	The documentation passes this review.
Outputs	An updated Users Manual.
Standards	IEEE 1063-2000 Standards for Software User Documentation, IEEE 1028-1997 Standard for Software Reviews.

Step 4: Test

Objectives	To perform regression testing on the proposed release.
Dependencies	This step depends on Step 3 of this process.
Responsibilities	Team members as assigned by the team leader shall perform the tests.
Inputs	The integrated code proposed for release and the system test package.
Entrance Criteria	Successful completion of Step 3 of this process.
Task Description	The tester or testers insure that the system test set is modified to includes testing of all the new features and bug fix in this proposed release. They shall also insure that the test set documentation reflects these updated tests. The entire test set is run on all supported platforms. The results are reviewed by the tester or testers to insure the proposed release still produces the same results or explained differences for all problems and platforms. The testing documentation and results shall be placed under configuration control.
Verification	The results of this step shall be verified in the next step.
Exit Criteria	Successful test set execution on all supported platforms.
Outputs	Test results and updated test set documentation.
Standards	IEEE 829-1998 Standard for Software Test Documentation.

Step 5: Approve

Objectives	To approve a proposed release for release.
Dependencies	This step depends on Step 4 of this process.
Responsibilities	The team approves an Intermediate Release, and the team and the BoD must both approve a Major Release.
Inputs	Documentation package for each new feature and bug fix, integrated code, and test results from Step 4 of this process.
Entrance Criteria	Successful completion of Step 4 of this process.
Task Description	For an Intermediate Release or a Major Release, the team reviews the documentation for the new feature(s) and bug fixes and the test documentation and results to insure that the release is ready. For a Major Release, the BoD must review and approve all changes to the user interface, both input and output. Also for a Major Release, both the BoD and the team must approve the proposed release for international release. The documentation of these reviews and their results shall be placed under configuration control.
Verification	The team leader verifies the approvals.
Exit Criteria	For an Intermediate Release, the team approves the release. For a Major Release, both the team and the BoD approve the release.
Outputs	Approved release and reviews documentation.
Standards	IEEE 1028-1997 Standards for Software Reviews.

Step 6: Release

Objectives	To execute the release of the product to users.
Dependencies	This step depends on Step 5 of this process.
Responsibilities	The Release Manager does the release.
Inputs	The approved code and documentation package.
Entrance Criteria	Successful completion of Step 5 of this process.
Task Description	For an Intermediate Release, the Release Manager places the approved code in a directory available to all users. He or she also distributes a memo announcing the release and its contents and the availability of the documentation for the new features and bug fixes. All items should already be under configuration control.

	For a Major Release, the Release Manager assembles the complete package of approved code, updated Users Manual, and updated test set and submits them with a memo describing the changes to RSICC for international distribution. All items should already be under configuration control.
Verification	The team leader verifies the completion of this step.
Exit Criteria	The approved package is released to users.
Outputs	The approved package available to users.
Standards	None

Step 7: Closed

Objectives	To close the process of releasing a version of a code.
Dependencies	This step depends on Step 6 of this process.
Responsibilities	The team leader is responsible for closing the process.
Inputs	The released package
Entrance Criteria	The successful completion of Step 6 of this process.
Task Description	Once the team leader has verified that the release is complete, that the notifications have been sent, and all items are under configuration control, the process may be closed.
Verification	None
Exit Criteria	Release complete and products under configuration control.
Outputs	None
Standards	None

III. OTHER PROCESS

Step 1: Reported

Objectives	To submit a proposal on an issue that does not directly affect the codes.
Dependencies	None

Responsibilities	Any team member may submit a proposal.
Inputs	Title, reference (if any), and description of issue.
Entrance Criteria	None
Task Description	Team member supplies information to define the proposal.
Verification	Team leader verifies that required information has been supplied.
Exit Criteria	Team leader accepts submission.
Outputs	Completed submission
Standards	ISO 9000-3:1997 Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software, IEEE/EIA 12207.*n*-1996 Software Life Cycle Processes, IEEE 1219-1998 Standard for Software Maintenance.

Step 2: Investigate

Objectives	To investigate the submitted proposal.
Dependencies	This step depends on Step 1 of this process.
Responsibilities	The team leader accepts, prioritizes, and assigns a developer to or rejects the proposal. The developer investigates the proposal.
Inputs	The submitted proposal
Entrance Criteria	Successful completion of Step 1 of this process.
Task Description	The team leader reviews the proposal and accepts it or rejects it. If it is accepted, a developer is assigned and a priority is selected. The results of this review are documented and placed under configuration control. The assigned developer investigates the proposal. The results of the investigation are documented and placed under configuration control. This is similar to Step 3: Requirements of the Features & Bugs Process.
Verification	Completion of this step will be verified in Step 5: Inspect, if not sooner.
Exit Criteria	The results of the investigation are documented by the developer.
Outputs	Documented review by team leader, documented investigation by developer.

Standards	None

Step 3: Correct

Objectives	To develop a "correction" for the proposal.
Dependencies	This step depends on Step 2 of this process.
Responsibilities	The developer is responsible for producing the "correction."
Inputs	The completed investigation from Step 2 of this process.
Entrance Criteria	Successful completion of this process.
Task Description	The developer creates a "correction" based on the results of the completed investigation. This "correction" may be code for support software or it may be a new version of a document. This is similar to the Design and Code steps of the Features & Bugs process. The "correction" is placed under configuration control.
Verification	The completion of this step will be verified in Step 5: Inspect, if not sooner.
Exit Criteria	A "correction" is completed by the developer.
Outputs	A "correction"
Standards	None

Step 4: Document

Objectives	To document the "correction."
Dependencies	This step depends on Step 3 of this process.
Responsibilities	The developer is responsible for completing the documentation.
Inputs	The "correction" from Step 3 of this process.
Entrance Criteria	Successful completion of Step 3 of this process.
Task Description	If the "correction" itself is not a document, the developer documents the "correction." The documentation is placed under configuration control.
Verification	The completion of this step will be verified in the next step, Inspect.
Exit Criteria	The documentation of the "correction" completed by the developer.
Outputs	The documentation of the "correction."

Standards	IEEE 1063-2000 Standards for Software User Documentation.

Step 5: Inspect

Objectives	To inspect the "correction" and its documentation.
Dependencies	This step depends on Step 4 of this process.
Responsibilities	The team performs the inspection.
Inputs	The "correction" from Step 3 of this process and the documentation from Step 4 of this process.
Entrance Criteria	Successful completion of Step 4 of this process.
Task Description	The team shall inspect the "correction" and its documentation to insure that it is complete and correct. If either fail this inspection, the process returns to the Investigate step and continues from there. The inspection and its results shall be documented and placed under configuration control.
Verification	The team leader shall verify the results of the inspection.
Exit Criteria	Successful completion of the inspection.
Outputs	Documentation of the inspection and its results.
Standards	IEEE 1028-1997 Standard for Software Reviews.

Step 6: Test

Objectives	To "test" the correction.
Dependencies	This step depends on Step 5 of this process.
Responsibilities	Team members or others assigned by the team leader shall "test" the "correction."
Inputs	The inspected "correction" and its documentation.
Entrance Criteria	Successful completion of Step 5 of this process.
Task Description	If the "correction" is code, for example a script, the tester(s) shall test it in a manner similar to the testing of features and bugs. The tests and their results shall be documented and placed under configuration control.

	If the "correction" is documentation, it shall be reviewed and edited by appropriate individuals. The edited "correction" shall be placed under configuration control. If the "correction" is something else, it shall be tested in an appropriate manner to insure it meets the requirements determined in Step 2: Investigation. The tests and results of the testing shall be documented and placed under configuration control.
Verification	Verification of this step shall be done in the next step.
Exit Criteria	Successful completion of the testing and documentation of the results.
Outputs	Testing and results documentation.
Standards	IEEE 829-1998 Standard for Software Test Documentation.

Step 7: Review

Objectives	To review the "correction," its documentation, and its testing.
Dependencies	This step depends on Step 6 of this process.
Responsibilities	The team shall perform this review.
Inputs	The "correction," its documentation, its test documentation and results.
Entrance Criteria	Successful completion of Step 6 of this process.
Task Description	The team or team leader shall review the entire package from submission through testing for completion. They shall also review the test documentation and results to insure that the "correction" meets requirements and has been completely and properly tested. If any part of the package fails this review, the process is returned to the Investigate step and continues from there. The review and its results are documented and placed under configuration control.
Verification	The team leader shall verify completion of this step.
Exit Criteria	Successful completion of this review.
Outputs	Documentation of the review and its results.
Standards	IEEE 1028-1997 Standard for Software Reviews.

Step 8: Closed

Objectives	To close the process on the submitted proposal.

Dependencies	This step depends on Step 7 of this process.
Responsibilities	The team leader is responsible for closing the process.
Inputs	The complete package for the proposed item.
Entrance Criteria	Successful completion of Step 7 of this process.
Task Description	Once the team leader has verified that the completed proposal has passed the review, that it has been released, if necessary, any necessary notifications sent, and all items are under configuration control, this process may be closed.
Verification	None
Exit Criteria	Successful completion of the review and release, if necessary, and all items under configuration control.
Outputs	None
Standards	None

IV. MEETINGS PROCESS

Step 1: Schedule

Objectives	To plan, schedule, and hold a review meeting.
Dependencies	This step depends on one of several steps from one of the other processes.
Responsibilities	For a BoD review, the BoD chair shall be responsible. For a team review, the team leader will be responsible.
Inputs	The item(s) to be reviewed.
Entrance Criteria	The item(s) are complete and ready for review.
Task Description	For a BoD review, the chair determines which items will be on the agenda. The items may include proposed new features, integrated and tested features ready for release, and proposed Major Releases. A team review will usually be a small subset of the team reviewing one or a few items. It may be an SRS or SDD, documentation, or test results. It may also be a feature or bug fix ready for release or a proposed Intermediate or Major Release. The team leader will assign the moderator and reviewers and what is to be reviewed.

The responsible person determines how long the review will take and how many people will be present. He or she selects a meeting site and a time to hold the meeting. All interested parties are notified about time, place, and agenda. Adequate advance notice and distributions of item(s) to be reviewed is necessary in order for the reviewers to have adequate time to prepare.

When the review convenes, the moderator shall inquire as to whether the reviewers have spent adequate time in preparation. If not enough reviewers are present, or they have not spent adequate time in preparation, or for a similar reason, the moderator adjourns the review, and the process returns to Step 2 of this process to reschedule the review.

If an adequate number of reviewers are present and they are prepared, the review continues. The secretary records all issues and concerns raised by the reviewers. When the review is complete, it is adjourned for the author to respond to the issues and concerns raised. If the review is not completed in the allotted time, the reviewers can decide to recess the review until a later date or to continue to complete the review. If it is decided to recess, this process returns to Step 2 for rescheduling the review. When the review is complete, the issues, concerns, and decisions are documented and placed under configuration control.

Verification	The team leader will verify this step is complete, including scheduling, adequate notice and preparation time are given, and the meeting is held. The moderator will verify that the review is complete and documented.
Exit Criteria	Documentation of the review and its results.
Outputs	List of issues and concerns of the reviewers, documentation of the review and its results.

Step 2: Actions

Objectives	To complete the response of the author to the issues and concerns of the reviewers.
Dependencies	This step depends on Step 1 of this process.
Responsibilities	The moderator and the author are responsible for completion of this step.
Inputs	The item(s) reviewed and the list of issues and concerns of the reviewers.
Entrance Criteria	Successful completion of Step 1 of this process.

Task Description	The author takes the list of issues and concerns and determines his or her response to each of them. If it was a document that was reviewed, the appropriate changes are made in the document. If it was code that was reviewed, the appropriate recoding is done. The author may decide that an explanation to the reviewer or no action may be the appropriate response. When the author completes his or her responses, they are sent to the moderator for review. The moderator may request further work on one or more responses. If necessary, the moderator may close this process and return the parent process to the appropriate previous step to accomplish the rework. When the moderator is satisfied with the responses, he or she may close this step or schedule another review, if the changes in the item(s) are significant. If the moderator decides to have another review, this process returns to Step 2 for scheduling the review and proceeds from there. The responses and modified item(s) shall be placed under configuration control.
Verification	The moderator shall verify that this step is complete.
Exit Criteria	Completed, acceptable responses to the issues and concerns of the reviewers.
Outputs	Modified item(s), documented responses from author.
Standards	IEEE 1028-1997 Standard for Software Reviews.

Step 3: Closed

Objectives	To close this review.
Dependencies	This step depends on Step 2 of this process.
Responsibilities	The team leader is responsible for closing the review.
Inputs	Modified item(s), documented responses from author.
Entrance Criteria	Successful completion of Step 2 of this process.
Task Description	The team leader reviews the documentation and results of the review, the author's responses, and the modified item(s). If the team leader is not satisfied, he or she consults with the moderator, and if necessary, returns the process to Step 1 to redo the review. When the team leader is satisfied, the review is closed.
Verification	None

Exit Criteria	Review documentation and results, author responses, and modified item(s) under configuration control.
Outputs	None
Standards	IEEE 1028-1997 Standard for Software Reviews.

V. WORKING PROCESS

Step 1: Active

Objectives	To work a task.
Dependencies	An issue in another issues group.
Responsibilities	The developer will be responsible.
Inputs	A task from another issue.
Entrance Criteria	There is a task in another issue that needs to be subdivided and tracked more closely.
Task Description	The developer works the task generating work products, as appropriate. Any work products generated are placed under configuration control.
Verification	Developer verifies task is complete.
Exit Criteria	Completed task
Outputs	Updated issue(s) and other products generated by task placed under configuration control.
Standards	None

Step 2: Closed

Objectives	To close this issue.
.	
Dependencies	This step depends on Step 1 of this process.
Responsibilities	The developer is responsible for closing the issue.
Inputs	Updated issue(s) and other products generated by task.
Entrance Criteria	Successful completion of Step 1 of this process.
Task Description	The developer closes the issue.

Verification	None
Exit Criteria	Issue is closed.
Outputs	None
Standards	None

APPENDIX E.

ROLES

I. ROLE DESCRIPTION

A. Anyone

Process	State	Action
Features & Bugs	Submitted	Submit new feature request of bug report.

B. Board of Directors

Process	State	Action
Features & Bugs	Submitted	Optional review of requested Features, document decision.
	Approve	Review entire Feature package especially user interface. Approve for release if satisfied.
Release	Approve	Review Major Release package, especially user interface, to make sure the release is ready for international distribution. Approve if it is ready.
Meetings	Schedule	For a BoD review, determine agenda and schedule review. Notify members of meeting and agenda. Allow sufficient time for reviewers to examine the material to be reviewed. Conduct review. Record all issues and concerns. Recess meeting if insufficient time to review all items or postpone them to another meeting. Promote to Action after meeting completed.
	Action	Work all action items resulting from meeting. BoD leader reviews completed action items to see if they require further BoD review and action. After all action items are acceptably completed, leader verifies completion and all items are under configuration management and response on all action items has been sent to BoD members. When all is complete, leader Closes review.

C. Developer

Process	State	Action
Features & Bugs	Requirements	Develop Software Requirements Specification (SRS), promote to Design when SRS completed.
	Design	Create design and write Software Design Document (SDD) Request review of SRS and SDD when SDD complete.
	Code	Develop code to comply with SDD, perform unit testing, promote to Document when coding and unit testing complete.
	Document	Document code, unit testing, and update Users Manual. Promote to Integrate when complete.
Other	Investigate	Investigate proposal. Promote to Correct when investigation is complete.
	Correct	Create a "correction." Promote to Document when "correction" completed.
	Document	If "correction" is not a document, document it. Promote to Inspect after documentation is complete.
Working	Active	Creates issue linking it to parent issue and works issue. When completed, documents activities and promotes issue to Closed.

D. Integrator

Process	State	Action
Features & Bugs	Integrate	Integrate reviewed code into baseline. Promote to Test when complete.

E. MC Team

Process	State	Action
Features & Bugs	Submitted	Submit new feature request of bug report.
	Design	Review and approve SRS and SDD, promote to Code if approved or to Requirements if rejected.

	Integrate	Line-by-line review of code and documentation including comparing with SDD. Demote to Requirements if doesn't pass review.
	Test	Update and perform regression test suite on integrated code in all supported environments. Promote to Approve when all tests passed.
	Approve	Review entire package including integration and testing results. Approve if complete and ready for release.
Release	Document	Reviews documentation of each included item. For an Intermediate Release, this documentation is change-pages. For a Major Release, this is updated Users Manual especially Chapters 2 and 3.
	Test	Perform regression test suite in all supported environments. Promote to Approve when all tests passed.
	Approve	Review documentation and test results. Approve Release if package complete and ready.
Other	Reported	Submit request on noncode issue.
	Inspect	Review "correction" and its documentation. Return process to Investigate if review failed, promote to Test if passed.
	Test	Test "correction."
	Review	Review entire "correction" package including test results.
Meetings	Schedule	Moderator schedules review and notifies members of the meeting and agenda. Allow sufficient time for reviewers to examine the material to be reviewed. Conduct review. Verify that reviews spent sufficient time reviewing material. Record all issues and concerns. Recess meeting if insufficient time to review all items or postpone them to another meeting. Promote to Action after meeting completed.
	Action	Work all action items resulting from meeting. Moderator reviews completed action items to see

		if they require further review and action. After all action items are acceptably completed, moderator verifies completion and all items are under configuration management and response on all action items has been sent to reviewers.

F. MC Team Leader

Process	**State**	**Action**
Features & Bugs	Submitted	Verified issue submitted. Reviews, approves, and prioritizes issue. Reviews BoD recommendations. Assigns Developer for approved items. Documents decision. Promotes to approved issues Requirements and rejected issues to Release.
	Document	Assign Integrator.
	Approve	Promote to Release after both team and BoD approve Feature or only team approves Bug fix.
	Finalize	Oversee release of approved Feature or Bug fix to users. This may include creating an Intermediate or Major release. Verify notifications of release or rejection have been distributed. Insure package including all documentation is complete and under configuration management. When process is complete, Close issue.
Release	Proposed	Concur with release proposal. Promote to Complete.
	Complete	Concur with list of items to be included. Promote to Document.
	Document	Promote to Test after team approves documentation.
	Approve	Promote to Release after both BoD and team approves Major Release or just team approves Intermediate release.
	Release	Promote to Closed after verifying release is completed and everything is under configuration management.
Other	Reported	Promote to Investigate when submission is complete.

	Investigate	Accept or Reject submission. Assign Developer and Priority if submission is accepted.
	Review	After team approval, verify package is complete and under configuration management. Promote to Closed when package complete.
Meetings	Schedule	For team review, determine what will be reviewed and who moderator and reviewers will be.
	Action	Reviews documentation and results of review. If not satisfied, returns review to Schedule. If satisfied, verifies all is under configuration management, then Closes review.

G. Release Manager

Process	**State**	**Action**
Release	Proposed	Propose new Intermediate or Major Release.
	Complete	Verifies each Feature and Bug fix proposed for inclusion in new release is complete and closed. When all are ready, freeze the baseline until release is complete.
	Release	Distributes as required, approved release with updated documentation. Sends notifications.

H. Software Quality Analyst

Process	**State**	**Action**
		There are no activities defined for this role in the SQA Plan. However, this role will oversee the execution of this plan.

I. Razor Administrator

Process	**State**	**Action**
		Razor was in use at the time the SQA Plan was written, so this role is not defined in it. The Razor Administrator will oversee the operation of the Razor databases used in the execution of this plan.

APPENDIX F.

WORK PRODUCTS

I. WORK PRODUCTS

A. Software Requirements Specification

Description: A Software Requirements Specification (SRS) documents essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces (IEEE 610.12-1990).

Resources used: Feature or enhancement request or bug report from user or team.

Created: Features & Bugs Process—Requirements step.

Reviewed: Features & Bugs Process—Design step.

Used: Features & Bugs Process—Design, Document, and Test steps.

Standards: IEEE 830-1998 Recommended Practice for Software Requirements Specifications.

B. Software Design Description

Description: A Software Design Description (SDD) is a representation of software system created to facilitate analysis, planning, implementation, and decision-making. The SDD is used as a medium for communicating software design information, and may be thought of as a blueprint or model of the software system (IEEE 610.12-1990).

Resources used: SRS

Created: Features & Bugs Process—Design step.

Reviewed: Features & Bugs Process—Design step.

Used: Features & Bugs Process—Coding, Document, and Inspect steps.

Standards: IEEE 1016-1998 Recommended Practice for Software Design Descriptions.

C. Code

Description: Code is computer instructions and data definitions expressed in a programming language or in a form output by an assembler, compiler, or other translator (IEEE 610.12-1990).

Resources used: SDD

Created: Features & Bugs Process—Code step; Other Process—Correct step.

.

Reviewed: Features & Bugs Process—Integrate step, Other Process—Inspect step.

.

Used: Features & Bugs Process—Integrate, Test, Approve, and Finalize steps; Release Process—Complete, Document, Test, Approve, and Release steps; Other Process—Inspect, Test, and Review steps; Meetings Process—Schedule and Actions steps.

Standards: Reference 13

D. Software Documentation

Description: Any written or pictorial information describing, defining, specifying, reporting, or certifying activities, requirements, procedures or results (IEEE 610.12-1990). This includes the SRS, SDD, User Manual, data model, data dictionary, and any other documentation necessary to describe the software, how it works, and how it is used.

Resources used: SRS, SDD, code.

Created: Features & Bugs Process—Documentation step; Release Process—Document step; Other Process—Document step.

Reviewed: Features & Bugs Process—Integration step; Release Process—Approve step; Other Process—Review step.

Used: Features & Bugs Process—Integration, Test, Approve, and Finalize steps; Release Process—Test, Approve, and Release steps; Other Process—Inspect, Test, and Review steps.

Standards: IEEE 1063-2000 Standards for Software User Documentation.

E. Software Test Plan

Description: A document describing the scope, approach, resources, and schedule of intended test activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning. (IEEE 610.12-1990).

Resources used: SRS, Users Manual.

Created: Features & Bugs Process–Code and Test steps; Release Process—Test step, Other Process—Test step.

Reviewed: Features & Bugs Process-Approve step; Release Process—Approver step; Other Process—Review step.

Used: Features & Bugs Process—Code and Test steps; Release Process—Test step, Other Process—Test step.

Standards: IEEE 1008-1987 (R1993) Standard for Software Unit Test.

F. Software Test Results

Description: This describes the conduct and results of testing carried out for a system or component (IEEE 610.12-1990).

Resources used: SRS, STP, test suite documentation.

Created: Features & Bugs Process—Code and Test steps; Release Process—Test step, Other Process—Test step.

Reviewed: Features & Bugs Process—Approve step; Release Process—Approve step; Other Process—Review step.

Used: Features & Bugs Process—Code and Test steps; Release Process—Test step, Other Process—Test step.

Standards: IEEE 829-1998 Standard for Software Test Documentation.

G. Review Results

Description: This included review meeting minutes, action items generated, action item responses, metrics generated, and any other documentation from the review and its results.

Resources used: Items to be reviewed.

Created: Meetings Process—Actions step.

Reviewed:	Meetings Process—Actions step.
Used:	Features & Bugs Process—Design, Integrate, and Approve steps; Release Process—Document and Approve steps; Other Process—Inspect and Review steps.
Standards:	IEEE 1028-1997 Standard for Software Reviews, IEEE 1012-1998 Standard for Software Verification and Validation.

APPENDIX G.

SOFTWARE VALIDATION AND VERIFICATION PLAN

I. PURPOSE

The purpose of this Software Verification and Validation Plan (SVVP) is to describe the verification and validation (V&V) activities of the Monte Carlo team used to improve the quality of the codes developed and maintained by members of the team. This includes, but is not limited to, MCNP.

The scope of this SVVP is the ongoing development and phases of the typical software life cycle. This plan supercedes SVVP in the previous MCNP SQA plan[3] and is written to reflect the use of a specific software-based tool, Razor™,[16] to control and monitor this software development process. The process itself and the implementation in Razor are described in detail in Ref. 4 that is extracted into Appendices B through F of the Monte Carlo Team SQA plan. Additional documents will be produced providing additional details for some of the process steps. This SVVP does not yet cover the V&V of the data libraries distributed and used with these codes. Such a plan is currently being developed. Also, a V&V plan covering the use of the code and the data together has not yet been written.

This SVVP is primarily for the team members who maintain and improve these codes. However, the audience is not only the members of the Monte Carlo team, but also its international user community and the management of ASCI, of which the Eolus Project is currently a part, as an indication of how the codes are maintained and improved. It will be used as a reference in assessments and audits by organizations external to this team and by users required to use software that satisfies specified V&V requirements. This is a living document, and is subject to ongoing revisions.

This plan is based on the requirements in IEEE Std 1012-1998.[G1] Also, ISO 9001:2000[7] was used in preparation of this plan.

™ Razor is a trademark of Tower Concepts, now a part of Visible Systems Corporation.

II. DEFINITIONS

See Appendix A.

III. V & V OVERVIEW

"Software V&V processes determine whether development products of a given activity conform to the requirements of that activity, and whether the software satisfies its intended use and user needs. This determination may include analysis, evaluation, review, inspection, assessment, and testing of software products and processes. V&V processes assess the software in context of the system, including the operational environment, hardware, interfacing software, operators, and users."[G 1]

V&V processes provide an objective assessment of software products and software development processes to demonstrate whether software requirements are correct, complete, accurate, consistent, and testable. Other objectives include facilitating early detection and correction of software errors, enhancing team and management insight into process and product risk, and supporting software life cycle processes to ensure compliance with program performance, schedule, and budget requirements.[G 1]

Verification and validation can mean different things depending on the viewpoint. Two different sets of definitions for these terms are included in Appendix A of the Monte Carlo Team SQA plan. The viewpoint used for the previous two paragraphs is the software engineering viewpoint. It describes verification as the activities performed to ensure that the outputs of a development process phase comply with the inputs of that phase, and validation as the activities performed to ensure that the outputs of the process meet the requirements stated at the start of the process. As Barry Boehm described V&V in 1979:[G2]

Validation: Are we building the right product?

Verification: Are we building the product right?

The computational physics viewpoint of V&V is different. It is focused on the correctness and applicability of the physics models and infrastructure used in developing the code (i.e., physics) rather than the development process. That viewpoint looks at whether the physics models are adequate to describe the real world and whether those models were coded correctly. This viewpoint is represented by the second set of definitions, from Ref. G3, for

validation and verification in Appendix A. Here the definitions are summarized in a more specific form as:

Validation: Are we solving the correct equations?

Verification: Are we solving the equations correctly?

A schematic drawing, adapted from Ref. G4, relating computational physics V&V activities to the "real world" is shown in Fig. G1. A model of nature is expressed in a theory that is represented by equations. Code verification is the evaluation of a computer simulation to ensure that the equations are properly implemented in the code. It includes comparison of calculated results with analytical solutions, regression test sets, and other codes. The computer simulation is validated by comparing its output, using any required physical data, with the results from experiments.

The relationship of experimental data to this drawing is not simple. Diagnostics that are part of experiments produce the measured data that can be compared to the results of computer simulations of the experiments. On the other hand, physical data may be used by the code to produce the computer simulation. This physical data is obtained from other experiments and is verified by comparison with theory. In this case, the verification and validation of the computer simulation are the verification and validation of the code and data as a combined set.

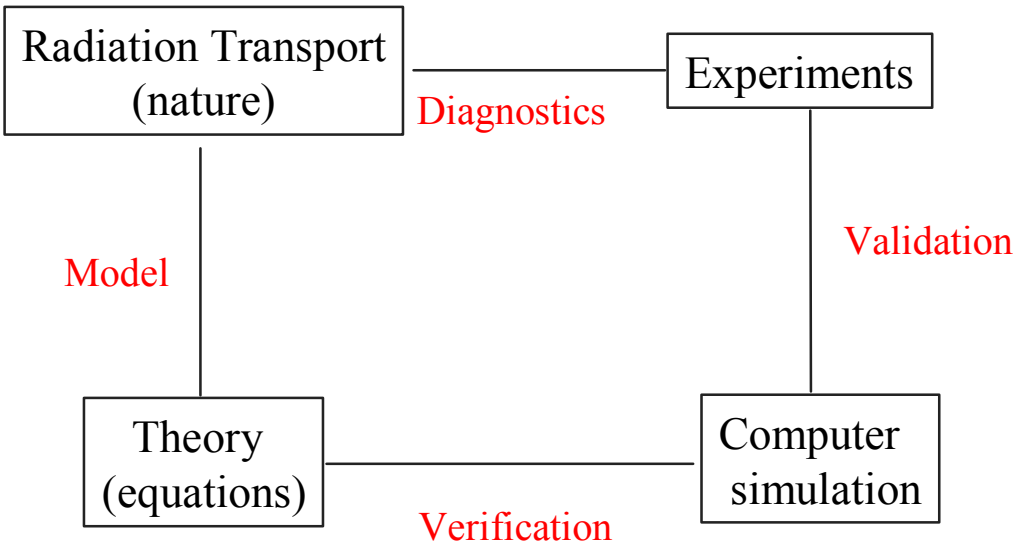

Figure G1. Relationship Between Nature and Computer Code

These computational physics definitions of V&V are used for MCNP. In this case, verification is activities that can be performed without regard to the quality of the nuclear data, and validation is performed for the combination of MCNP and the best available nuclear data.

In the computational physics view of V&V, software engineering processes are part of the conversion of theory to computer simulations. On the other hand, software engineering would consider these definitions of V&V as part of the software development process validation activities.

Because both the software engineering and computational physics viewpoints are relevant to the codes developed by the Monte Carlo team, the V&V process will be detailed from each viewpoint.

- The software engineering viewpoint will be used in Section G.IV.A describing process V&V.
- The computational physics viewpoint will be used in Section G.IV.B describing code product V&V.

The remainder of this SVVP is divided into four sections. Section G.IV describes both V&V processes used by the Monte Carlo team. The reporting, administrative, and documentation requirements of the process tasks will be described in Sections G.V through G.VII.

The scheduling of many of the software V&V activities is described in Appendices B through D in the Monte Carlo Team SQA plan. The scheduling of the remaining activities is included in Section G.IV.

Software integrity levels[G1] or similar risk assessment criteria are frequently used as part of verification and validation to denote a range of software criticality values necessary to maintain risks within acceptable limits; namely, how critical is the software and how severe are the consequences of a failure in the software. These levels are based on the intended use and application of the software. Ref. G1 uses the following software integrity levels and describes them in more detail. Documents are under development for risk assessment in ASCI programs at LANL and for more general risk assessment at LANL.

Criticality	Description	Level
High	Selected function affects critical performance of a system	4
Major	Selected function affects important system performance	3
Moderate	Selected function affects system performance, but workaround strategies can be implemented to compensate for loss of performance	2
Low	Selected function has noticeable effect on system performance but only creates inconvenience to the user if the function does not perform in accordance with requirements.	1

As the consequences of software failure increase, the breadth and formality of verification and validation activities increases. Because MCNP and other codes developed by the Monte Carlo team may be used in many applications of which the team is not aware, the assignment of software integrity levels for those applications will not be done. However, an assessment of the software for applications that the team is aware of within LANL will be done as part of assessments of those applications by higher levels of program management. The V&V activities required by the level(s) selected will be determined at that time.

The resources for process V&V and code product verification for Monte Carlo team codes are the members of the team. They verify the completion of each step and of the entire processing of an issue. The Monte Carlo team leader will perform many of the V&V activities and is responsible for ensuring that all V&V activities that are team responsibilities are completed. The validation of code product results will be done by separate ASCI and other outside teams to be named by those organizations.

The tools, techniques, and methods to perform the process V&V tasks are described in Appendices B through D in the Monte Carlo Team SQA Plan. The tools, techniques, and methods necessary to perform the additional computational physics V&V tasks included in Section G.IV will be described there.

IV. V&V PROCESS

According to Ref. G1, the standard SVVP includes descriptions and tasks involved in the management, acquisition, supply, development, operation, and maintenance processes of a software life cycle. The acquisition and supply processes do not apply to Monte Carlo team software because they are the providers of the maintenance and upgrade activities of their codes. Code developed by contractors to the team will be treated the same as code developed by the team. The remaining life cycle processes are integrated into the development process

described in the Monte Carlo Team SQA Plan Appendices B through D and therefore should be considered as a unit. The contents of individual parts of a V&V overview, organization, master schedule, resource summary, etc., are included in the text of Sections G.IV.A and G.IV.B and are not described in separate subsections.

Reference G1 contains a table indicating the minimum V&V tasks assigned to each software integrity level. This is a very long, detailed list so that it can be applied to any project including very large ones. For the size of software projects covered by this V&V plan, the list is vast overkill and needs to be greatly tailored. Although this SQA Plan contains most of the required activities, a table that maps this plan's activities to those listed in Ref. G1 will not be produced at this time. A primary reason is that software integrity level of the software covered by this plan has not been determined.

The greatest area of noncompliance in this V&V plan with that overall guidance is in the area of software testing. Although a number of test suites have been developed by the Monte Carlo team, a plan guiding the preparation on contents of the suites and the individual tests contained in them has not been prepared. This shortcoming is being corrected.

A. Software Development Process V&V

The various process step verification activities are mentioned in the process description in Appendix B of the SQA plan, and are listed as part of the detail in the description of each process step in Appendix D of that plan. The subprocesses in the Monte Carlo Team software development process are: 1) Features, Enhancements, and Bugs; 2) Release; 3) Reviews; 4) Other; and 5) Working. In the detailed description, the verification action in each step describes the V&V actions needed to assure the completion of that step and the role or roles that perform those actions. The overall V&V of the products and process is done by the team leader as part of preparing to close an issue. Summaries of these actions for the first four subprocesses are given in Tables G1 through G4. Because the Working subprocess is a simple open-closed working process, no V&V activities are needed.

TABLE G1 FEATURES, ENHANCEMENTS, AND BUGS SUBPROCESS STEP VERIFICATION ACTIONS	
Subprocess Step	Step Verification Activity
Submitted	Team Leader verifies required information has been supplied
Requirements	SRS is reviewed and accepted by the team at the design review
Design	Team approval of SRS and SDD
Code	Verification shall be done in this step or the following Integrate step
Document	Documentation will be validated in the Integrate step
Integrate	Successful review by the team and integration into the base code
Test	Results from this step shall be verified in the next step
Approve	Team leader shall verify the completion of this step
Closed	None

TABLE G2 RELEASE SUBPROCESS STEP VERIFICATION ACTIONS	
Subprocess Step	Step Verification Activity
Propose	Team leader concurs
Complete	Team leader shall concur with the list of included items
Document	Team leader verifies that this documentation is complete and passed review
Test	Results of this step shall be verified in next step
Approve	Team leader verifies approvals
Release	Team Leader verifies completion of this step
Closed	None

TABLE G3 OTHER SUBPROCESS STEP VERIFICATION ACTIONS	
Subprocess Step	Step Verification
Reported	Team leader verifies that required information has been supplied
Investigate	Completion of this step will be verified in Inspect step, if not sooner
Correct	Completion of this step will be verified in Inspect step, if not sooner
Document	Completion of this step will be verified in Inspect step
Inspect	Team leader shall verify the results of the inspection
Test	Verification of this step shall be done in the next step
Review	Team leader verifies completion of this step
Closed	None

TABLE G4 REVIEWS SUBPROCESS STEP VERIFICATION ACTIONS	
Subprocess Step	Step Verification
Schedule	Team leader verifies the scheduling and that adequate notice and preparation time are given
Actions	Moderator will verify that the review and action items are complete
Closed	None

Software verification by the designated developer is part of the tasks in many of the steps, and the Reviews subprocess is a verification activity itself. In the Features, Enhancements, and Bugs subprocess, team (peer) reviews are part of the Design, Integrate, and Approve steps. The BoD performs a review of proposed new features and the integration of those features. This review may include a review of the V&V related to that feature. The testing of the changes in the Test step is also a part of the verification process. In the Release subprocess, the team performs reviews in the Document and Approve steps, and the BoD reviews major releases before they are released to the world user community. The team performs reviews in the Inspect and Review steps of the Other subprocess; there are no BoD reviews in this subprocess.

The validation of the process results takes place in the Approval or Review step of the subprocesses. Here, the Monte Carlo team reviews the products of each subprocess step especially the results from running the regression test suite and any additional tests on the developed code and the results from other product V&V activities. For new features integrated

into the code, the BoD also reviews the process results including changes to the user interface. Because there is no defined schedule for the completion of any step or subprocess, the V&V activities are scheduled as part of the completion of a step before the issue can be advanced, or are specific steps in a subprocess such as the Approve step.

B. Code Product V&V

Product verification includes a number of activities. These are included in Fig G2. After a new feature, enhancement, or bug fix is coded, the code is verified by comparison to the results obtained from other sources of information such as analytical solutions to the model being tested, benchmarks, or results from other codes. Examples of these are found in Refs. 16-17 and G5-G8. A large variety of benchmarks has been defined by various user communities, and they evaluate the codes developed by the Monte Carlo team against them. Also, problems are defined such that they can be calculated by codes supported by the Monte Carlo team and by codes from other sources. This verification is repeated after integration to ensure that no negative effects have occurred as a result of the integration.

Unit testing is typically done as part of the code development stage and is verified when the code is integrated into the baseline version. These unit tests can be retained and included in additional test sets that can be used for regression and verification testing.

For the Monte Carlo team, the use of the regression test suite is the primary means of code version-to-version verification. This suite is run whenever new features, enhancements, or bug fixes are integrated into the code. All tests exercising unaffected code should produce results identical to those obtained before the integration. Identical in this case means that: 1) the same number of random numbers are used, implying identical particle random walks; and 2) the answers agree to six significant figures (with few exceptions due to round-off errors such as in small differences between two large, approximately equal numbers). This test suite is expanded as new features, enhancements, and bug fixes are included in the code. A description of the regression test suite and the procedures for updating it including adding tests for new code as it is integrated is in the Software Test Plan (to be written). A goal is to provide as complete coverage of code execution as possible.

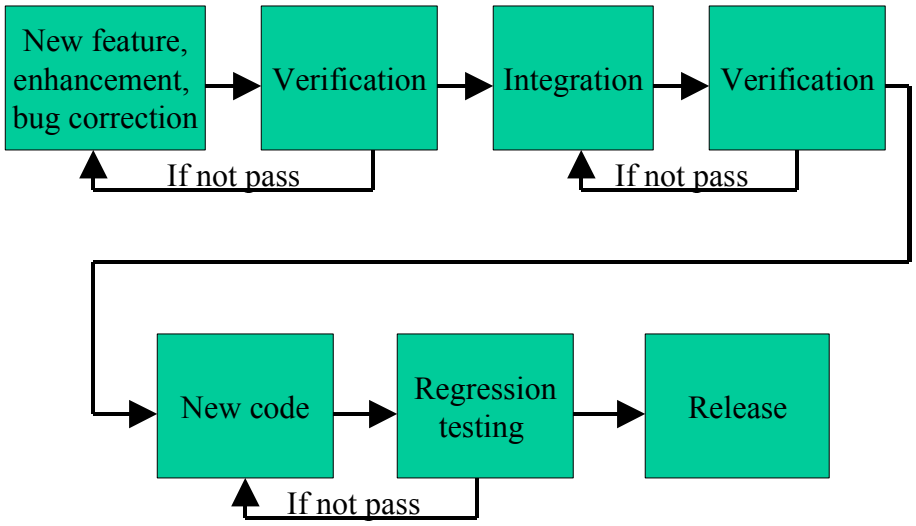

Figure G2. Code Product Verification

Code product validation is done by comparing results from "real world" experiments with those results calculated by a code and any associated databases for a model description of the experiment. These comparisons indicate how well the code and data calculate the results of the experiment and how well the employed model and data represent the real world. Some of these comparisons are included in Refs. G9 and G10. A validated code can also be used to predict, within its range of applicability and with observed biases and expected statistical variation, the results of an experiment to be performed. These validation activities are not done by the Monte Carlo team and are not part of this plan.

In addition to the product V&V activities performed by the Monte Carlo team, many user organizations perform their own V&V on the codes to ensure the reliability of a code to their specific applications. The results of these additional tests are frequently published in open literature.

Because Monte Carlo team software development life cycle is ongoing, everything is continually being tested. Additionally, at each release, a code is tested again on all supported platforms. In the future, test sets will include the team's published tests (benchmarks) that will be used as additional verification activities.

V. V&V REPORTING REQUIREMENTS

A. Process V&V

The reporting requirements are described in the narrative process description in Appendix B of the SQA plan and in the detailed process step descriptions in Appendix D of that plan. They include new feature and enhancement requests and bug reports that are part of that subprocess and the review and testing results from all subprocesses. Additional reports may be defined as this software development process evolves.

B. Product V&V

The template copies of the results from running the regression test suite on release versions of the codes are stored in the configuration management database. These results are used to evaluate the effects of code changes. The results of these evaluations are reported to the team and are published as research notes, meeting summaries, and/or peer-reviewed journal articles.

The results from comparisons with analytical solutions, benchmarks, or results from other codes will be reported in Los Alamos Research Notes and may also be published as Los Alamos National Laboratory reports or in open literature.

VI. V&V ADMINISTRATIVE REQUIREMENTS

A. Process V&V

The administrative requirements are described in the narrative process description in Appendix B of the SQA plan and in the detailed process step descriptions in Appendix D of that plan. In particular, code anomalies are reported as issues in the Features, Enhancements, and Bugs subprocess. The control procedures for the software development process are embodied in the use of Razor to implement the process and the tracking included in that product.

B. Product V&V

The administrative requirements of product V&V are included in the administrative requirements for process V&V and are described there.

VII. V&V DOCUMENTATION REQUIREMENTS

A. Process V&V

The software development process V&V documentation is the test documentation, including the test plans, cases, procedures, and results. The test cases for previous versions of MCNP have been documented in Refs. 16 and 17. An update to those will be published. The template copies of the results from running the regression test suite on release versions of the codes are included with the major releases and for all releases are to be stored in the configuration management database. The software test plan and test procedures still need to be written.

B. Product V&V

The documentation of results from product V&V will be reported in Los Alamos Research Notes and also may be published as Los Alamos National Laboratory reports.

REFERENCES

These references are additional references specifically used for this appendix. See Section XIV of the SQA plan for references used throughout this document.

G1. IEEE Std 1012-1998, "IEEE Standard for Verification and Validation," Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017-2394 (March 9, 1998).

G2. Barry W. Boehm, "Software Engineering: R & D Trends and Defense Needs," Research Directions in Software Technology (P. Wegner, ed.) MIT Press, Cambridge, MA (1979).

G3. AIAA Guide G-077-1998, "Guide for the Verification and Validation of Computational Fluid Dynamics Simulations," American Institute of Aeronautics and Astronautics, 1801 Alexander Bell Drive, Suite 500, Reston, VA 20191-4344 (1998).

G4. Cindy Zoldi, "Fluid Instabilities: A Computational and Experimental Analysis of a Shock-Accelerated Heavy-Gas Cylinder," 2001 Science Day, Los Alamos National Laboratory (February 13, 2001).

G5. Avneet Sood, R. Arthur Forster, and Donald K. Parsons, "Analytical Benchmark Test Set for Criticality Code Verification," Los Alamos National Laboratory Report LA–13511, Los Alamos, NM (July 1999). http://lib-www.lanl.gov/la-pubs/00460062.pdf

G6. John D. Court, John S. Hendricks, and Stephanie C. Frankle, "MCNP ENDF/B-VI Validation: Infinite Media Comparisons of ENDF/B-VI and ENDF/B-V," Los Alamos National Laboratory Report, LA–12887 (December 1994). http://lib-www.lanl.gov/la-pubs/00287187.pdf

G7. John D. Court, Ronald C. Brockhoff, and John S. Hendricks, "Lawrence Livermore Pulsed Sphere Benchmark Analysis of MCNP ENDF/B-VI," Los Alamos National Laboratory Report, LA–12885 (December 1994). http://lib-www.lanl.gov/la-pubs/00287083.pdf

G8. J. D. Court and J. S. Hendricks, "Benchmark Analysis of MCNP ENDF/B-VI Iron," Los Alamos National Laboratory Report, LA–12884 (December 1994). http://lib-www.lanl.gov/la-pubs/00287071.pdf

G9. D. J. Whalen, D. A. Cardon, J. L. Uhle, and J. S. Hendricks, "MCNP: Neutron Benchmark Problems," Los Alamos National Laboratory report, LA–12212 (November 1991). http://lib-www.lanl.gov/la-pubs/00285970.pdf

G10. D. J. Whalen, D. E. Hollowell, and J. S. Hendricks, "MCNP: Photon Benchmark Problems," Los Alamos National Laboratory report, LA–12196 (September 1991). http://lib-www.lanl.gov/la-pubs/00285969.pdf and http://lib-www.lanl.gov/la-pubs/00194077.pdf

Los Alamos NM 87545